Bachelor's Thesis

Towards Automated Theorem Proving in the CloG Proof System

Han Meerholz (S3987299)
First Supervisor: Helle Hvid Hansen
Second Supervisor: Revantha Ramanayake

August 15, 2022

Abstract

Game logic is a logic system used for reasoning about determined two-player games. Recently, Enqvist et al (2019) proved the completeness of game logic by making use of a circular proof system named CloG. Since CloG is a cut-free sequent system, it should be suitable for Automated Theorem Proving. To explore these automated proof search capabilities, and create an implementation of the CloG proof system, a tool has been developed in the programming language of Rascal.

This thesis covers the implementation of this tool, and discusses the results of several strategies applied in this automated proof search. We found that resulting proof shapes and execution times varied based on the inputted sequent and applied strategy.

Contents

1 Introduction and Motivation

1.1 Background and Example

Game logic is an area of logic first introduced in 1985 [17]. It is an abstract logic that can describe the strategic abilities of players in two-player determined games. It describes how a player can have a strategy to ensure that, after the game in question is played, a certain property or combination of properties holds. To make this more concrete, let us consider the prisoner's dilemma, which can be seen as a simple, determined, two-player game.

The prisoner's dilemma [20] describes a scenario in which two criminals are about to be imprisoned. There is not enough evidence for conviction on a principle charge, only for a lesser charge. Now both criminals are told in isolation that they can choose to testify against the other, such that they go free, while the other goes to prison for three years. If, however, both prisoners testify against one another, they both serve two years. And if both remain silent, they only serve one year. The game can be summarized in a pay-off matrix, where the higher the prison sentence, the lower the pay-off. To achieve this, we make the number of years served in prison negative. See table 1.1.

		Prisoner B	
		Stays Silent	Testifies
Prisoner A	Stays Silent	-1, -1	-3, 0
	Testifies	0, -3	-2, -2

Table 1: Pay-off matrix for the Prisoner's Dilemma. The first number specifies pay-off (negative number of years served) for prisoner A, and the second number specifies the pay-off for prisoner B.

If prisoner A would remain silent, depending on prisoner B's choice, prisoner A would either either serve 1 or 3 years in prison. We can label the four outcomes p_0, p_1, p_2, p_3, where p_n means prisoner A will serve n years in prison. Then we can say that prisoner A can ensure either p_1 or p_3 holds. In game logic syntax this would denoted $\langle\gamma\rangle(p_1 \vee p_3)$, which means that prisoner A has a strategy to ensure $p_1 \vee p_3$ holds after game γ has been played. In this case, γ refers to the Prisoner's Dilemma. Note that even though prisoner A has a strategy to ensure one of these outcomes, they do not have a strategy to ensure only p_1 holds, and they do not have a strategy to ensure only p_3 holds. Thus, it is not the case that $\langle\gamma\rangle(p_1 \vee p_3) \rightarrow \langle\gamma\rangle(p_1)$: this game is not disjunctive. However, it does work the other way around: if player 1 can ensure either p_1 or p_3 holds, then they can also ensure p_1, p_2, or p_3 holds. This is called *monotonicity.*

This is just one of the interesting properties of game logic. Another one is *determinacy*: if prisoner A does not have a strategy to ensure an outcome, this means prisoner B has a strategy to ensure the negation of this outcome. Since we know that prisoner A cannot ensure that they themselves will go free, this means prisoner B can ensure that they will not go free (by testifying against prisoner A). This can be denoted as $\neg\langle\gamma\rangle p_0 \rightarrow \langle\gamma^d\rangle\neg p_0$, where γ^d corresponds to the game γ, but with the roles for prisoner A and B reversed. I.e. any choice made by prisoner A in game γ will be made by prisoner B in game γ^d and vice versa.

1.2 Research Question and Approach

Game logic is abstract and can be used to reason about two-player determined games other than the Prisoner's Dilemma. Moreover, it can reason about player's strategies in more complex games built from different kinds of atomic games.

A proof system for game logic, CloG was introduced in [7]. This proof system is a *sequent calculus* (see section 2.2), which is *cut-free* (also see section 2.2). Because this proof system does not contain rules for which we have to guess a certain lemma or invariant, it is suitable for automated theorem proving. This led to the research question: "How can we carry out Automated Theorem Proving for a practical implementation of the CloG proof system?"

In this thesis, a tool will be described that can automatically generate proofs for *sequents* (see section 2.2) in the CloG proof system. We want to ensure that this tool is correct (i.e. the proofs generated are correct), sound (i.e. it cannot generate proofs for invalid sequents), complete (i.e. it can generate a proof for each valid sequent), and efficient (if possible, minimize the search space, execution time, and resulting proof sizes).

There are some challenges in developing such a proof search algorithm. Because CloG is a circular proof system (see section 2.3), we need to account for detecting both desired and undesired cycles. And for a more efficient proof search (minimizing at least search space and execution time), we need certain proof strategies.

As this subject matter is very much theoretical and falls under foundational research, the direct practical applications of the tool described in this thesis are limited. People who would benefit the most from a tool like this are researchers of fields related to game logic and automated theorem proving. Not only can such a tool be beneficial for formal verification of interacting systems and agents, but an implementation of the CloG proof system may also pave the way to the creation of (automated) theorem provers for different logics and proof systems, applying similar strategies. Lastly, this tool may be combined with proof transformation tools. One of these tools is the one developed in [22], which can transform a CloG proof into a proof in the G proof system (another proof system for game logic introduced in [7]). Another proof transformation tool has been developed by a fellow bachelor student this year, and transforms from the G proof system to the first game logic proof system developed, Par, introduced in [17]. Combining our automated proof search tool with these proof transformation tools allows us to automatically find proofs in the CloG, G, and Par proof systems.

Relevant topics in Computing Science are thus Theoretical Computing Science, Formal Verification, and Automated Theorem Proving.

2 Theoretical Background

2.1 Game Logic

Game logic is a *non-normal modal* logic introduced in [17], which is used to describe the strategic abilities of two players (often named Angel and Demon) to obtain certain outcomes in a game.

Modal logics are a type of logics that extend the basic propositional logic by adding modal operators [16]. These modal operators can take on different meanings depending on the context, but in general, we have the $\Diamond$ and $\Box$ operators, where $\Box\varphi = \neg\Diamond\neg\varphi$. The modal operators are understood as follows: $\Diamond\varphi$ means that from the current world, a world in which φ is true is accessible, and $\Box\varphi$ means φ is true in all accessible worlds from the current world [23].

Game logic is *non-normal.* A *normal* logic can be modeled by Kripke models [23], but a *non-normal* logic lacks certain properties of a normal logic, and can be described by neighborhood semantics, a generalization of Kripke semantics, instead. Since *monotonicity* holds for Game Logic (if $p \to q$ is valid, then if Angel has a strategy in game γ to achieve p, Angel must also have a strategy in game γ to achieve q. I.e. $\langle\gamma\rangle p \to \langle\gamma\rangle q$ is is valid), we can use *monotone* neighborhood models. The semantics for these neighborhood models are further explained in section 2.1.2.

In game logic, we have game modalities, and a modal formula $\langle\gamma\rangle\varphi$ means that Angel has a strategy to force an outcome such that φ holds after playing game γ. Its dual, $[\gamma]\,\varphi$, denotes that Angel does not have a strategy to force an outcome where φ does not hold after playing game γ, which is equivalent to saying that Demon has a strategy to ensure that φ holds after playing game. This is because games described in game logic are determined (if Angel "wins", Demon "loses" and vice versa) [19].

2.1.1 Syntax

The following is the original game logic syntax as defined in [17], denoted $\mathcal{L}_{\mathsf{Par}}$. Here, φ is any game logic proposition, Φ_0 is the set of propositional atoms, γ is a game, and Γ_0 is the set of atomic games.

Definition 2.1 (Game Logic Syntax $\mathcal{L}_{\mathsf{Par}}$)**.**

$$\begin{aligned}\mathcal{L}_{\mathsf{Par}} \ni \varphi &\coloneqq p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\gamma\rangle\varphi \\ \mathcal{G}_{\mathsf{Par}} \ni \gamma &\coloneqq g \mid \gamma\,;\gamma \mid \gamma \sqcup \gamma \mid \gamma^* \mid \gamma^d \mid \varphi?\end{aligned}$$

where $p \in \Phi_0$ and $g \in \Gamma_0$.

Relevant for this thesis, we use a slightly different syntax, the so called *normal form language* $\mathcal{L}_{\mathsf{NF}}$ as defined in [7]. In this normal form, we only allow the negation and dual operators on atomic propositions and atomic games respectively, and we include the demonic operators (explained below):

Definition 2.2 (Game Logic Syntax $\mathcal{L}_{\mathsf{NF}}$)**.**

$$\begin{aligned}\mathcal{L}_{\mathsf{NF}} \ni \varphi &\coloneqq p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle\gamma\rangle\varphi \\ \mathcal{G}_{\mathsf{NF}} \ni \gamma &\coloneqq g \mid g^d \mid \gamma\,;\gamma \mid \gamma \sqcup \gamma \mid \gamma \sqcap \gamma \mid \gamma^* \mid \gamma^\times \mid \varphi? \mid \varphi!\end{aligned}$$

where $p \in \Phi_0$ and $g \in \Gamma_0$.

We will explain what each operator in game logic means and what it means for such an operator to appear in a propositional formula or game modality.

The $\neg$, $\vee$, and $\wedge$ symbols are simply negation, disjunction, and conjunction, as in propositional logic. The formula $\neg\varphi$ is true just in case φ is false. The formula $\varphi \vee \psi$ is true just in case φ is true, or ψ is true, or both are true, and $\varphi \wedge \psi$ is true just in case φ is true and ψ is true. As mentioned before, $\langle\gamma\rangle\varphi$ means that Angel has a strategy to ensure φ holds after playing game γ. A dual game, γ^d is the game γ where the roles of Angel and Demon are reversed. Thus, $\langle\gamma^d\rangle\varphi$ means that Demon has a strategy to ensure φ holds after playing game γ. Game concatenation, $\gamma\,;\delta$, describes a game where first γ is played, and then δ. Thus, $\langle\gamma\,;\delta\rangle\varphi$ means Angel has a strategy to ensure φ holds after first playing γ, and then playing δ. Angelic choice, $\gamma \sqcup \delta$, describes a game in which Angel can choose whether to play γ or δ. Thus, $\langle\gamma \sqcup \delta\rangle\varphi$

means Angel has a strategy to ensure φ holds after either playing γ or δ, where Angel can choose which of these is played. Demonic choice, $\gamma \sqcap \delta$, is the opposite, equivalent to $(\gamma^d \sqcup \delta^d)^d$, Demon can choose whether to play γ or δ. Thus, $\langle \gamma \sqcap \delta \rangle \varphi$ means Angel has a strategy to ensure φ holds both in case game γ, and in case game δ is played. The angelic test, $\varphi?$, corresponds to the game in which you check whether φ is true, if not, then Angel loses. Thus, $\langle \varphi? \rangle \psi$ simply means Angel has a strategy to ensure φ holds, and they have a strategy to ensure ψ holds. The demonic test, $\varphi!$, corresponds to the game in which Angel wins if φ is true. It is equivalent to $(\neg\varphi?)^d$. Thus, $\langle \varphi! \rangle \psi$ means Angel has a strategy to ensure φ holds, or they have a strategy to ensure ψ holds. Finally, angelic iteration, γ^*, corresponds to the game in which Angel decides how often to play the game γ: zero or more times. Thus, $\langle \gamma^* \rangle \varphi$ means Angel has a strategy to ensure φ holds after playing γ a specific number of times (or not at all). Demonic iteration, $\gamma^\times$, is angelic iteration's dual operation and means that Demon can choose how many times to play the game. This is equivalent to $((\gamma^d)^*)^d$. Thus, $\langle \gamma^\times \rangle \varphi$ means Angel has a strategy to ensure φ holds no matter how often γ is played.

2.1.2 Semantics

For a monotone modal logic like game logic, to account for two-player games, monotone neighborhood structures are used [18].

A *neighborhood model* consists of a set of *worlds* or *states*. Each state s has a set of neighborhoods, and $\Box\varphi$ is true in s just in case $[\![\varphi]\!]$ *is* one of the neighborhoods of s. Here, $[\![\varphi]\!]$ denotes the *truth set* of φ, or all the worlds in which φ is true [16].

A Game Model is a neighborbood model, which we can formally denote as follows:

Definition 2.3 (Game Model). *A game model $\mathbb{S}$ is a tuple $(S,\ E,\ V)$, such that S is a set of states, E assigns an effectivity function E_g to each atomic game $g \in G_0$. V is a valuation function, such that $V(p)$ returns a set of worlds for which the propositional atom p is true.*

An *effectivity function* then is defined as follows:

Definition 2.4 (Effectivity Function). *E_γ is the effectivity function for the game γ, with signature $\mathcal{P}(S) \to \mathcal{P}(S)$. I.e. this function maps from a set of states to another set of states. This effectivity function must be monotone: $X \subseteq Y \implies E_\gamma(X) \subseteq E_\gamma(Y)$. For a subset $X \in \mathcal{P}(S)$, $E_\gamma(X)$ is the set of starting states in which γ is played, such that Angel can ensure the outcome of the game played to be in X. Alternatively phrased, $E_\gamma(X)$ is the set of states at which Angel is effective for X in γ.*

What follows is the definition of the truth set or *meaning* of a formula φ in the game model $\mathbb{S}$, $[\![\varphi]\!]_\mathbb{S}$, and the effectivity function for complex games.

Definition 2.5 (Game Logic Semantic Valuation).

$$\begin{array}{lcl}
[\![p]\!]^\mathbb{S} & := & V(p) \\
[\![\neg p]\!]^\mathbb{S} & := & S \setminus p \\
[\![\varphi \vee \psi]\!]^\mathbb{S} & := & [\![\varphi]\!]^\mathbb{S} \cup [\![\psi]\!]^\mathbb{S} \\
[\![\varphi \wedge \psi]\!]^\mathbb{S} & := & [\![\varphi]\!]^\mathbb{S} \cap [\![\psi]\!]^\mathbb{S} \\
[\![\langle\gamma\rangle\varphi]\!]^\mathbb{S} & := & E_\gamma([\![\varphi]\!]^\mathbb{S}) \\
E_{(\gamma^d)}(X) & := & S \setminus E_\gamma(S \setminus X) \\
E_{\gamma\,;\,\delta}(X) & := & E_\gamma(E_\delta(X)) \\
E_{\gamma\sqcup\delta}(X) & := & E_\gamma(X) \cup E_\delta(X) \\
E_{\gamma\sqcap\delta}(X) & := & E_\gamma(X) \cap E_\delta(X) \\
E_{(\gamma^*)}(X) & := & \mathsf{lfp}\, Y.\, X \cup E_\gamma(Y) \\
E_{(\gamma^\times)}(X) & := & \mathsf{gfp}\, Y.\, X \cap E_\gamma(Y) \\
E_{(\varphi?)}(X) & := & [\![\varphi]\!]^\mathbb{S} \cap X \\
E_{(\varphi!)}(X) & := & [\![\varphi]\!]^\mathbb{S} \cup X
\end{array}$$

Here, $\setminus$, $\cup$, and $\cap$ stand for *set difference* ($A \setminus B$ means all elements that are in A, but not in B), *set union* ($A \cup B$ means all elements that are in either A or B (or both)), and *set intersection* ($A \cap B$ means all elements that are in both A and in B), respectively.

The lfp and gfp operators are *least fixpoint* and *greatest fixpoint* operators, respectively.

For either fixpoint operator, we construct a function with signature $\mathcal{P}(S) \to \mathcal{P}(S)$. For the least fixpoint formula, our initial input is the empty set, and for the greatest fixpoint formula, our initial input is the full set of states S. For the notation $lfp\, Y.f(Y)$ or $gfp\, Y.f(Y)$, we simply execute $f(Y)$ with Y being this initial input. If the output of the function is equal to Y, we have reached our fixpoint, and the result of the fixpoint operator is this Y. Otherwise, we take the output of the function, and feed it into the same function again, and we continue this process until $f(Y) = Y$.

2.2 Sequent Calculi

The proof systems relevant in this thesis are sequent calculi. These were originally introduced by Gentzen. An English translation of his original paper can be found in [8]. A sequent calculus or sequent system consists of sequents of the form $A_1, ..., A_n \vdash B_1, ..., B_m$, which can be read as $A_1 \wedge ... \wedge A_n \to B_1 \vee ... \vee B_m$. A sequent calculus contains axioms and inference rules. Axioms are always true, and can be derived from nothing; take for example the identity axiom ($\dfrac{}{A \vdash A}\, id$). Inference rules are derivations from one set of sequents to another, such as "and introduction on the right" ($\dfrac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\, R\wedge$) [14, 21].

Since for implication, $A \to B \equiv \neg A \vee B$, we can rewrite $A_1 \wedge ... \wedge A_n \to B_1 \vee ... \vee B_m$ as $\neg(A_1 \wedge ... \wedge A_n) \vee B_1 \vee ... \vee B_m$. If we now apply the De Morgan law $\neg(A \wedge B) \equiv \neg A \vee \neg B$, we can rewrite it further as $\neg A_1 \vee ... \vee \neg A_n \vee B_1 \vee ... \vee B_m$, which is equivalent to the one-sided sequent $\vdash \neg A_1, ..., \neg A_n, B_1, ..., B_m$.

This means we can rewrite any sequent as a one sided sequent, and likewise, we can speak of one-sided sequent systems, where all the rules only use one-sided sequents and there is no need for a distinction between left and right rules [21].

2.3 Circular Proofs

A proof in a sequent calculus usually consists of a finite tree with the conclusion at the root. Game logic, however, is a fixpoint logic. Just like modal mu-calculus, a more well known logic, widely used in certain areas of computing science [2], game logic has monotone modal operators.

To calculate these fixpoints, we often need to repeatedly *unfold* branches ad infinitum, resulting in an infinite number of branches in the proof tree. A proof system allowing for infinite branches is called an infinitary proof system [4]. To represent an infinite branch, we can also create a loop or back-link to a lower node, resulting in a cyclic proof. This often eliminates the need of having to guess what certain invariants or lemmas would have to be [10]. Circular proofs can be used in something as simple as First Order Logic [3], but have also been used to allow for a cut-free sequent calculus for μ-calculus [1]. Moreover, circular proofs are implicitly used in the once again cut-free sequent calculus CloG, introduced in [7], and thus are an essential part of the tool described in this thesis.

2.4 Automated Theorem Proving

Automated Theorem Proving (ATP) is an established field of Computing Science and Artificial Intelligence. It is concerned with mechanizing the proving process of a logic system [13]. While not as popular as the numerical applications of computers, ever since the 1950s, people have tried to implement the principles of logic into an automated machine. Over the years, theorem provers became more efficient, and started being applied to program verification and computer security [12].

Many earlier ATP systems revolved mostly around First-Order Logic, but ATP has been applied to many different logics, among which modal logics [9] and implementations of sequent systems [15]. Recently, there have also been examples of ATP applied to cyclic proof systems, such as in [5].

Proof search in sequent systems often starts at the bottom conclusion, then the rules are applied in reverse until axioms are reached on all branches, at which point the conclusion will have been proven. Many basic sequent systems include a cut rule, ($\dfrac{\Gamma \vdash A, \Delta \qquad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}\, cut$). It turns out that in many of these systems, such as the LJ and LK sequent systems introduced in [8], but also in more modern sequent systems such as [1], the cut rule can be left out, but we can still derive the exact same conclusions in the sequent system. Often, we desire this *cut elimination* in a sequent calculus [21]. If cut elimination can be achieved, it often makes it easier to apply automated theorem proving to the sequent calculus. This is because the premise of the cut rule contains a formula, A, which does not occur in the conclusion. Thus, applying the cut rule in reverse means having to guess whatever this A would have to be, and this could theoretically be any formula in the logic. A cut-free system eliminates the need to guess this formula.

3 The CloG Proof System

$$\dfrac{\Phi, \varphi^{\mathsf{a}}, \psi^{\mathsf{a}}}{\Phi, (\varphi \lor \psi)^{\mathsf{a}}}\lor \qquad \dfrac{\Phi, \varphi^{\mathsf{a}} \qquad \Phi, \psi^{\mathsf{a}}}{\Phi, (\varphi \land \psi)^{\mathsf{a}}}\land \qquad \dfrac{}{p^{\varepsilon}, (\neg p)^{\varepsilon}}\,\mathsf{Ax1} \qquad \dfrac{\Phi}{\Phi, \varphi^{\mathsf{a}}}\,\mathsf{weak}$$

$$\dfrac{\Phi, (\langle\gamma\rangle\varphi \lor \langle\delta\rangle\varphi)^{\mathsf{a}}}{\Phi, (\langle\gamma \sqcup \delta\rangle\varphi)^{\mathsf{a}}}\sqcup \qquad \dfrac{\Phi, (\langle\gamma\rangle\varphi \land \langle\delta\rangle\varphi)^{\mathsf{a}}}{\Phi, (\langle\gamma \sqcap \delta\rangle\varphi)^{\mathsf{a}}}\sqcap \qquad \dfrac{\varphi^{\mathsf{a}}, \psi^{\mathsf{b}}}{(\langle g\rangle\varphi)^{\mathsf{a}}, (\langle g^d\rangle\psi)^{\mathsf{b}}}\,\mathsf{mod}_m \qquad \dfrac{\Phi, \varphi^{\mathsf{ab}},}{\Phi, \varphi^{\mathsf{axb}}}\,\mathsf{exp}$$

$$\dfrac{\Phi, (\psi \land \varphi)^{\mathsf{a}}}{\Phi, (\langle\psi?\rangle\varphi)^{\mathsf{a}}}\,? \qquad \dfrac{\Phi, (\psi \lor \varphi)^{\mathsf{a}}}{\Phi, (\langle\psi!\rangle\varphi)^{\mathsf{a}}}\,! \qquad \dfrac{\Phi, (\langle\gamma\rangle\langle\delta\rangle\varphi)^{\mathsf{a}}}{\Phi, (\langle\gamma\,;\delta\rangle\varphi)^{\mathsf{a}}}\,;$$

$$(\mathsf{a} \preccurlyeq \langle\gamma^*\rangle\varphi)\ \dfrac{\Phi, (\varphi \lor \langle\gamma\rangle\langle\gamma^*\rangle\varphi)^{\mathsf{a}}}{\Phi, (\langle\gamma^*\rangle\varphi)^{\mathsf{a}}}\,* \qquad (\mathsf{a} \preccurlyeq \langle\gamma^\times\rangle\varphi)\ \dfrac{\Phi, (\varphi \land \langle\gamma\rangle\langle\gamma^\times\rangle\varphi)^{\mathsf{a}}}{\Phi, (\langle\gamma^\times\rangle\varphi)^{\mathsf{a}}}\,\times$$

$$(\mathsf{a} \preccurlyeq \mathsf{x} \in N_{\langle\gamma^\times\rangle\varphi}, \mathsf{x} \notin \Phi, \mathsf{a})\ \begin{array}{c} [\Phi, \langle\gamma^\times\rangle\varphi^{\mathsf{ax}}]^{\mathsf{x}} \\ \vdots \\ \dfrac{\Phi, (\varphi \land \langle\gamma\rangle\langle\gamma^\times\rangle\varphi)^{\mathsf{ax}}}{\Phi, (\langle\gamma^\times\rangle\varphi)^{\mathsf{a}}} \end{array}\,\mathsf{clo}_{\mathsf{x}}$$

Figure 1: The axiom and rules of the system CloG. In the side condition of clo_x, "$\mathsf{x} \notin \Phi, \mathsf{a}$" means that x does not occur in Φ or a.

Definition 3.1 (Fixpoint Formula)**.** *A fixpoint formula is either a greatest fixpoint formula of the form $\langle\gamma^\times\rangle\varphi$ or a least fixpoint formula of the form $\langle\gamma^*\rangle\varphi$.*

The CloG proof system, presented in figure 1, is a sequent calculus for game logic, where sequents can simply be read as disjunctions of formulae. Each formula in CloG is annotated. This annotation, or *label*, is a sequence of names. Each of these names is associated with a unique *greatest fixpoint formula* (see definition 3.1). This means for each greatest fixpoint formula φ, there is a set of names N_φ. The full set of names N then is the union of the sets of names N_φ of each greatest fixpoint formula.

An empty label is denoted by ε. For convenience sake, we will refer to annotated formulae simply as "formulae" from this point on. Most of the rules are fairly straight forward. Each of them (except Ax1 and clo) propagate their label from the premise(s) to the conclusion.

A proof in the CloG proof system has a tree structure, where each of the nodes is an inference rule or axiom application. The leaves of the tree are either an application of the Ax1 axiom, or a discharged assumption for the clo rule.

What follows is an explanation of each of the CloG rules and the axiom:

The axiom, Ax1, states that an atomic proposition or its negation is always valid. The weak rule states that if a sequent is valid, then that sequent plus an additional formula is valid also. With the exp rule, we can see that if a formula with a label is valid, that same formula with an extra name in the label is also valid.

The $\lor$ rule states that we can replace two formulae in the sequent by a disjunction of these two formulae, which can be understood through the fact that the sequent is read disjunctively.

The $\land$ rule is the only binary rule application and states that if we have two sequents with the same side formulae, and one differing formula, then we can replace them by one sequent with the same side formulae and the conjunction of these two differing formulae.

The $\sqcup$ rule is derived from the equivalence $\langle\gamma\rangle\varphi \lor \langle\delta\rangle\varphi \leftrightarrow \langle\gamma \sqcup \delta\rangle\varphi$. Saying "Angel has a strategy to ensure φ holds after playing game γ, or Angel has a strategy to ensure φ holds after playing game δ" is the same as saying "Angel has a strategy to ensure φ holds after choosing to play either game γ or game δ, thus after playing the game $\gamma \sqcup \delta$". The same argument can be made for the $\sqcap$ rule, but with conjunction instead of disjunction.

The ; rule is similarly derived from the equivalence $\langle\gamma\rangle\langle\delta\rangle\varphi \leftrightarrow \langle\gamma\,;\delta\rangle\varphi$. Saying "Angel has a strategy to ensure, after playing game γ, it holds that Angel has a strategy to ensure φ holds after playing game δ" is the same as saying "Angel has a strategy to ensure φ holds after first playing game γ and then game δ, thus after playing the game $\gamma\,;\delta$".

The ? rule is derived from the equivalence $\psi \wedge \varphi \leftrightarrow \langle\psi?\rangle\varphi$. Saying that "$\psi$ and φ are valid" is the same as saying "Angel has a strategy to ensure φ holds after playing the game where, if ψ does not hold, Angel loses". This is because. if ψ is not valid, Angel immediately loses, and if ψ is valid, $\langle\psi?\rangle\varphi$ is only valid if φ is valid as well. A similar argument can be made for the ! rule, where if ψ is valid, Angel wins, thus $\langle\psi!\rangle\varphi$ is valid. If ψ is not valid, $\langle\psi!\rangle\varphi$ is only valid if φ is valid.

The mod_m rule can be understood through Game Logic's monotonicity. Assume the proposition $\psi \to \varphi$ is valid. Now, if Angel has a strategy to ensure ψ holds after playing a game g, since $\psi \to \varphi$, Angel also has a strategy to ensure φ holds after playing this same game. We can write this as $\psi \to \varphi \Rightarrow \langle g\rangle\psi \to \langle g\rangle\varphi$. The mod_m rule is just a variation of this rule. We can rewrite $\psi \to \varphi$ as $\neg\psi \vee \varphi$, and we can rewrite $\langle g\rangle\psi \to \langle g\rangle\varphi$ as $\neg\langle g\rangle\psi \vee \langle g\rangle\varphi$, which is equivalent to $\langle g^d\rangle\neg\psi \vee \langle g\rangle\varphi$. And if we now just replace $\neg\psi$ by ψ, we obtain our mod_m rule: $\varphi \vee \psi \Rightarrow \langle g\rangle\varphi \vee \langle g^d\rangle\psi$.

The $*$ rule is derived from the equivalence $\varphi \vee \langle\gamma\rangle\langle\gamma^*\rangle\varphi \leftrightarrow \langle\gamma^*\rangle\varphi$. Saying that "$\varphi$ holds now or Angel has a strategy to ensure φ holds after playing γ at least once" is the same as saying "Angel has a strategy to ensure φ holds after playing game γ any number of times (including zero)". This is the case since, if φ already holds, Angel can choose to play game γ 0 times, thus they have a strategy to ensure φ holds by choosing to play game γ 0 or more times. The $\times$ rule works similarly: "φ holds now, and Angel has a strategy to ensure φ would continue to hold if γ is played at least once more, no matter how often." is the same as saying "Angel has a strategy to ensure φ holds regardless of how often γ is played."

To understand the clo rules and the side conditions for the $*$, $\times$, and clo rules, we need to understand how the $\preccurlyeq$ symbol appertains to the partial ordering on fixpoint formulae.

Definition 3.2 (Ordering on Fixpoint Formulae)**.** *For $\circ, \dagger \in \{*, \times\}$: $\langle\gamma^\circ\rangle\varphi \prec \langle\delta^\dagger\rangle\psi$ if $\delta^\dagger \lhd \gamma^\circ$, where $\lhd$ indicates a subterm relation. $\gamma \lhd \delta$ just in case γ is a subterm of δ, thus γ appears somewhere in δ. Furthermore, $\langle\gamma^\circ\rangle\varphi \preccurlyeq \langle\delta^\dagger\rangle\psi \quad$ iff $\quad \delta^\dagger \unlhd \gamma^\circ$. This $\unlhd$ operator is simply defined as $\gamma \unlhd \delta \equiv (\gamma \lhd \delta$ or $\gamma = \delta)$.*

Now, we can also extend this fixpoint partial ordering to the names referring to the fixpoint formulae:

Definition 3.3 (Ordering on Names)**.** *For two names $x_0 \in N_\varphi, x_1 \in N_\psi$, $x_0 \preccurlyeq x_1 \quad$ iff $\quad \varphi \preccurlyeq \psi$.*

A similar definition can be given for the comparison between a name and a fixpoint formula. We can also apply this ordering to entire labels:

Definition 3.4 (Ordering on Labels)**.** *Given a label a and fixpoint formula φ, $a \preccurlyeq \varphi \quad$ iff $\quad$ for each name $x_i \in a$, $x_i \preccurlyeq \varphi$.*

We can say that applying the $*$, $\times$, and clo rules *unfolds* the fixpoint formulae they are applied to. The clo rule also adds a name to the formula it is applied to (and its second side condition ensures that this name does not already appear in the sequent). If $\Phi, (\langle\gamma^\times\rangle\varphi)^{\mathsf{a}x}$ appears somewhere above the node where the clo rule that introduced x was applied, the branch can be closed. This appearance serves as an assumption. It is possible for this assumption to appear on multiple branches of the proof. All of these occurrences of the assumption $(\Phi, (\langle\gamma^\times\rangle\varphi)^{\mathsf{a}x})$ are said to be *discharged* by the clo rule application for the name x. Assuming the other branches in the proof are closed as well (by the $\mathsf{Ax1}$ axiom or another discharged assumption), we have formed a circular proof.

The remaining side condition of the $*$, $\times$, and clo rules ensures that between a closure application and its discharged assumption, no fixpoint formulae of higher *priority* are unfolded, where priority refers to the fixpoint ordering (i.e. if $\varphi \prec \psi$, ψ is of higher priority than φ).

4 Requirements

What follows is a list of requirements for the automated theorem prover tool, following a variation of the MoSCoW method [6], with sections for "must have", "should have", and "could have" requirements respectively, followed by a section on non-functional requirements.

4.1 Must Have

This subsection covers the essential requirements for the tool that will be developed. Without these requirements, the tool will not work as intended.

R-1.1 There must be some way for a user to input a CloG sequent for which to generate a proof.

R-1.2 The tool must be able to apply automated proof search to the inputted sequent.

R-1.3 If a proof could be found, the tool must be able to output the found proof.

R-1.4 If no proof could be found, the tool must also inform the user.

R-1.5 The tool must be sound; i.e. if a sequent is not valid, no proof should be found for that sequent.

4.2 Should Have

This subsection covers other important requirements for the usability of the tool.

R-2.1 The tool should be complete; i.e. if a sequent is valid, the tool should be able to output a proof for that sequent.

R-2.2 The tool should have a way for the user to specify the order in which the rules are applied in the proof search algorithm.

4.3 Could Have

This subsection covers requirements that would be nice to have and add to the usability of the tool, but are non-essential.

R-3.1 The tool could have some way of indicating which formulae in the sequents in the proof are active, i.e. to which formulae rules are applied, for readability.

R-3.2 The tool could have some additional heuristics and methods for automatic proof search.

R-3.3 The tool could allow for live user-interaction, where a user can choose which rule to apply to the current sequent at every step.

R-3.4 The tool could have more user-interaction and a user-friendly UI.

4.4 Non-Functional Requirements

This subsection covers requirements that do not relate to the functionality of the tool itself, but rather to the reliability, efficiency, and maintainability of the tool.

R-4.1 The tool should be tested properly to ensure no blatant bugs are present.

R-4.2 The tool should not be unnecessarily slow or inefficient.

R-4.3 The tool should have proper documentation and commenting.

R-4.4 The tool should not have unnecessarily complicated or convoluted code.

5 Approach & Strategies

5.1 Depth-First Search

Let us first give a high-level overview of the manner in which we could apply Automated Theorem Proving to the CloG proof system, and general strategies therein.

The naive approach to finding a proof in the CloG proof system would be a simple depth-first search. We start with a sequent that we want to prove. Now, if we can apply the axiom, or if we can match a discharged assumption, we can close the current branch, and assuming the other branches are correct, we have found a proof. Otherwise, we try to apply one of the rules to the sequent, and try to find a proof for the resulting sequent or sequents. If we fail to find a proof for such a sequent, we can try to apply the next rule, or often, we first try to apply the same rule, but applied to a different formula in the sequent (as most of the rules are applied to a specific formula in the sequent). Only after we have tried to apply each of the rules in each possible way (to each formula in the sequent), and no proof was found for any of the resulting sequents or pairs of sequents (in case the $\wedge$ rule as applied), can we indicate that no proof was found. This way, we will have explored the full proof search space.

5.2 Challenges

There are several things to account for and improve upon for this naive depth-first search.

5.2.1 Cycle Detection

Desired Cycles

In order to discharge the sequent associated with a clo rule application, we need to find this same sequent (with the addition of a name to the fixpoint formula to which the closure rule was applied) somewhere above the proof node where the closure rule was applied. This can be seen from the way the closure rule is defined in the CloG system:

$$\begin{array}{c} [\Phi, \langle\gamma^{\times}\rangle\varphi^{\mathsf{ax}}]^{\mathsf{x}} \\ \vdots \\ (\mathsf{a} \preccurlyeq \mathsf{x} \in N_{\langle\gamma^{\times}\rangle\varphi}, \mathsf{x} \notin \Phi, \mathsf{a})\ \dfrac{\Phi, (\varphi \wedge \langle\gamma\rangle\langle\gamma^{\times}\rangle\varphi)^{\mathsf{ax}}}{\Phi, (\langle\gamma^{\times}\rangle\varphi)^{\mathsf{a}}}\ \mathsf{clo}_{\mathsf{x}} \end{array}$$

Thus, in order to close a branch with a closure sequent discharge, we can do the following. We can look at the names n of all the formulae in the current sequent, and for each n, retrieve the sequent that was the conclusion of the clo rule application where n was introduced. If one of these sequents is the same sequent as the current sequent, but where the fixpoint formula associated with n is not annotated with n itself, we have found the discharged assumption for the clo rule application and we can close the branch.

Furthermore, when trying to apply a $*$, $\times$, or clo rule to a formula, we need to make sure its side condition holds. For this, we once again need to check the fixpoint formulae associated with the names in the label of the formula to which one of these rules is applied.

Hence, our proof search algorithm needs some way of keeping track of not only the greatest fixpoint formulae associated with the names present in the labels of the formulae in the current sequent, but also the sequents in which these greatest fixpoint formulae appear. We can do this by passing a map along at each step of our proof search, that contains the greatest fixpoint formula and associated sequent for each of the names of the formulae in our current sequent. As for how this is implemented in the developed tool, this is explained later in section 6.2.1.

Undesired Cycles

Unfortunately, not all cycles that can occur within a proof search like this are desired and can be used to close a branch. Take for instance the sequent containing a single formula $\langle a^{**}\rangle p^{\varepsilon}$. To this sequent, we can apply the $*$ rule, then the $\vee$ rule, then the $*$ rule again, and keep alternating, resulting in the following proof:

$$
\begin{array}{cl}
\vdots & \\
p^{\varepsilon}, \langle a^{**}\rangle p^{\varepsilon}, \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle p^{\varepsilon} & \\ \hline
p^{\varepsilon}, \langle a^{**}\rangle p \vee \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle p^{\varepsilon}, \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle p^{\varepsilon} & \vee \\ \hline
p^{\varepsilon}, \langle a^{*}\rangle\langle a^{**}\rangle p^{\varepsilon}, \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle p^{\varepsilon} & * \\ \hline
p^{\varepsilon}, p \vee \langle a^{*}\rangle\langle a^{**}\rangle p^{\varepsilon}, \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle p^{\varepsilon} & \vee \\ \hline
p^{\varepsilon}, \langle a^{**}\rangle p^{\varepsilon}, \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle p^{\varepsilon} & * \\ \hline
p^{\varepsilon}, \langle a^{**}\rangle p \vee \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle p^{\varepsilon} & \vee \\ \hline
p^{\varepsilon}, \langle a^{*}\rangle\langle a^{**}\rangle p^{\varepsilon} & * \\ \hline
p \vee \langle a^{*}\rangle\langle a^{**}\rangle p^{\varepsilon} & \vee \\ \hline
\langle a^{**}\rangle p^{\varepsilon} & *
\end{array}
$$

As can be seen from this example, after twice a $*$ rule and $\vee$ rule application, we obtain a sequent, which is a superset of the original sequent. And after another twice a $*$ rule and $\vee$ rule application, not allowing for duplicate formulae in the sequent, we obtain the exact same sequent again. A naive depth-first search trying to find a proof for this sequent will get stuck in infinite recursion.

To prevent these kinds of undesired cycles, we can either check whether the current sequent is a superset of an earlier saved sequent, or is exactly the same sequent. Also, we need to be careful about the labels. If Φ, φ^a shows up somewhere above $\Phi, \varphi^{\varepsilon}$ and we cannot close the branch with a closure sequent discharge, then we still have an undesired cycle. Other cycles that may or may not show up may include Φ, φ^b above Φ, φ^a, or $\Phi, \varphi^a, \varphi^b$ above Φ, φ^a.

We need to either prevent any of these cycles to occur, or if they do occur, detect them, and backtrack to an earlier point in the proof to continue the proof search. Algorithm 6 in section 6.2.4 shows how this undesired cycle detection is implemented for this project.

5.2.2 Search Space

In order to make a proof search more efficient, we can try to restrict the *Search Space*. We can reduce the number of different proofs we have to check in order to find a proof or determine there is no proof for a sequent.

Weakening and Expanding Rule Ordering

The weak rule can be applied (in reverse) to any sequent with more than one formula, and the exp rule can be applied to any sequent with at least one formula containing a non-empty label. We can greatly reduce the search space by restricting when we can apply these weak and exp rules.

Definition 5.1 (Simple Rules)**.** *Let us call all logical rules (not the structural rules* weak *and* exp*) that are local (i.e. do not require information from other parts of the proof) applied to a single formula within the sequent, simple rules. Thus, the simple rules for the* CloG *proof system are the following: the $\vee$, $\wedge$, $\sqcup$, $\sqcap$, ;, ?, !, $*$, and $\times$ rules.*

Applying weak in reverse just removes one formula in the disjunction, to which none of these simple rules can then be applied. Thus, we can always first apply these simple rules before applying the weak rule, essentially moving the weak rule up. The same can be said for the exp rule. The simple rules merely propagate the label of the formulae they are applied to. We can first apply these simple rules before the exp rule, and still obtain the same sequent as we would obtain if we were to apply the simple rules after the exp rule.

The only places where we require the weak and exp rules to be applied *before* other rules, are to match the conclusion of the Ax1 axiom, or the one for the mod_m rule, or to match the assumption for a clo rule. To match the Ax1 axiom or mod_m rule premise, or the clo rule assumption, we already know the sequent to weaken to. For instance, if we have a sequent $p^{x_0}, q^{\varepsilon}, \neg p^{\varepsilon}$, we can check for any pairs of atomic propositions, and find out we want to obtain the sequent $p^{\varepsilon}, \neg p^{\varepsilon}$, which we can do through weakening and expanding rules.

Other Local Rule Ordering

In order to further reduce the search space, we need to realize that for all of the simple rules in the CloG system, the order in which they are applied does not matter. Take for instance a sequent like $(p \vee q)^\varepsilon, (r \wedge s)^\varepsilon$. If we first apply the $\vee$ rule to the first formula, and then the $\wedge$ rule to the second formula, we obtain two sequents $p^\varepsilon, q^\varepsilon, r^\varepsilon$ and $p^\varepsilon, q^\varepsilon, s^\varepsilon$, but these same sequents are obtained by first applying the $\wedge$ rule and then the $\vee$ rule. Thus, if we have a sequent for which we can either apply the $\vee$ rule, or the $\wedge$ rule, if we have tried applying the $\vee$ rule before the $\wedge$ rule and no proof was found, we do not need to try applying the $\wedge$ rule before the $\vee$ rule as well. This also holds for applying the same rule to two different formulae in the sequent ($(p \vee q)^\varepsilon, (r \vee s)^\varepsilon$ becomes $p^\varepsilon, q^\varepsilon, r^\varepsilon, s^\varepsilon$, no matter whether the $\vee$ rule is applied to the first or second formula first). Of course, there can be nested sequents $((p \vee q) \wedge r)^\varepsilon$, but in this case we cannot apply the $\vee$ rule first regardless, thus there is only one possibility for rule application ordering.

It turns out being able to swap rule applications, but still reach the same sequent, holds for almost all of the rules in CloG. Thus, generally, if a rule can be applied, and no proof is found for the resulting sequent, there is no reason to backtrack. There are exceptions, however. Namely, for the mod_m rule, because it requires weakening. Take for instance a sequent like $\langle a \rangle p^\varepsilon, \langle a^d \rangle r^\varepsilon, \langle a^d \rangle \neg p^\varepsilon$. As described in section 5.2.2, if we want to apply the mod_m rule, we can try apply weakening rules first in order to match the mod_m conclusion. Thus, we can apply the mod_m rule to the first two formulae in the sequent, by applying the weak rule on the last formula first, such that we end up with a sequent $p^\varepsilon, r^\varepsilon$, which is not valid. However, we could still apply the mod_m to the first and third formula in the sequent, by applying weak to the second one, such that we end up with $p^\varepsilon, \neg p^\varepsilon$, to which we can apply the axiom Ax1. We can give a similar example for a sequent like $p \vee \neg p^\varepsilon, \langle a \rangle q^\varepsilon, \langle a^d \rangle r^\varepsilon$, where, if we try applying the mod_m rule first, by applying weak to the first formula, we do not find a proof ($q^\varepsilon, r^\varepsilon$), whereas if we apply the $\vee$ rule first, we can apply weak on the last two formulae, and match the Ax1 conclusion ($p^\varepsilon, \neg p^\varepsilon$). Thus, for the mod_m rule, we *do* want to backtrack to before it was applied to see if there are other formulae to which we can apply the mod_m, or if we want apply other rules first instead.

Another rule of which the application might not be able to be freely exchanged, is the clo rule. If we were to switch a closure rule application with any other rule application, the sequent to which the closure rule is applied will be different, thus the assumption which will have to be matched further up in the proof in order to discharge the sequent, will be different as well. In order to get the same result, the proof must change around that assumption discharge as well then. While it might not be possible to apply the clo rule in any order, we can "push it up". Formally:

Lemma 5.1. *If at some nodes in a CloG proof, a simple rule application (see definition 5.1) appears directly above a* clo *rule application, we can interchange these two rule applications, and still construct a valid proof.*

Proof. We can prove this lemma by distinguishing between several cases for which local rule application can occur above the clo rule application. Let us first consider a simple unary rule inference. As an example, we will highlight a proof transformation for a $\vee$ rule application occurring directly above a clo rule application, but the same process can be applied to the $\sqcup$, $\sqcap$, ;, ?, !, $*$, and $\times$ rules.

$$
\begin{array}{c}
[\Phi, (\langle \gamma^\times \rangle \varphi)^{a,x}, (\psi \vee \omega)^b]^x \\
\vdots \\
\dfrac{\dfrac{\Phi, (\varphi \wedge \langle \gamma \rangle \langle \gamma^\times \rangle \varphi)^{a,x}, (\psi)^b, (\omega)^b}{\Phi, (\varphi \wedge \langle \gamma \rangle \langle \gamma^\times \rangle \varphi)^{a,x}, (\psi \vee \omega)^b}\,\vee}{\Phi, (\langle \gamma^\times \rangle \varphi)^{a}, (\psi \vee \omega)^b}\,\mathrm{clo}_x
\end{array}
\quad\Longrightarrow\quad
\begin{array}{c}
\dfrac{[\Phi, (\langle \gamma^\times \rangle \varphi)^{a,x}, (\psi)^b, (\omega)^b]^x}{\Phi, (\langle \gamma^\times \rangle \varphi)^{a,x}, (\psi \vee \omega)^b}\,\vee \\
\vdots \\
\dfrac{\dfrac{\Phi, (\varphi \wedge \langle \gamma \rangle \langle \gamma^\times \rangle \varphi)^{a,x}, (\psi)^b, (\omega)^b}{\Phi, (\langle \gamma^\times \rangle \varphi)^{a}, (\psi)^b, (\omega)^b}\,\mathrm{clo}_x}{\Phi, (\langle \gamma^\times \rangle \varphi)^{a}, (\psi \vee \omega)^b}\,\vee
\end{array}
$$

From the CloG proof on the left, we know we can derive $\Phi, (\langle \gamma^\times \rangle \varphi)^{a,x}, (\psi \vee \omega)^b$ from $\Phi, (\varphi \wedge \langle \gamma \rangle \langle \gamma^\times \rangle \varphi)^{a,x}, (\psi)^b, (\omega)^b$. If we now switch around the $\vee$ and clo rule application, we still end up with the sequent $\Phi, (\varphi \wedge \langle \gamma \rangle \langle \gamma^\times \rangle \varphi)^{a,x}, (\psi)^b, (\omega)^b$, from which we can yet again derive $\Phi, (\langle \gamma^\times \rangle \varphi)^{a,x}, (\psi \vee \omega)^b$, but to match the assumption of the clo rule applied after the $\vee$ rule, we need to apply the $\vee$ rule once more.

A more complicated proof transformation is necessary for the case where a $\wedge$ rule application occurs directly above the clo rule:

$$\dfrac{\dfrac{\begin{array}{c}[\Phi, (\langle\gamma^\times\rangle\varphi)^{a,x}, (\psi \wedge \omega)^b]^x\\ \vdots\\ \Phi, (\varphi \wedge \langle\gamma\rangle\langle\gamma^\times\rangle\varphi)^{a,x}, (\psi)^b\end{array} \qquad \begin{array}{c}[\Phi, (\langle\gamma^\times\rangle\varphi)^{a,x}, (\psi \wedge \omega)^b]^x\\ \vdots\\ \Phi, (\varphi \wedge \langle\gamma\rangle\langle\gamma^\times\rangle\varphi)^{a,x}, (\omega)^b\end{array}}{\Phi, (\varphi \wedge \langle\gamma\rangle\langle\gamma^\times\rangle\varphi)^{a,x}, (\psi \wedge \omega)^b}\,\wedge}{\Phi, (\langle\gamma^\times\rangle\varphi)^{a}, (\psi \wedge \omega)^b}\,\mathrm{clo}_x$$

$$\Longrightarrow$$

$$\dfrac{\begin{array}{c}\dfrac{[\Phi, (\langle\gamma^\times\rangle\varphi)^{a,x}, (\psi)^b]^x \qquad \begin{array}{c}\dfrac{\Phi, (\langle\gamma^\times\rangle\varphi)^{a,x,y}, (\psi)^b]^x \qquad \Phi, (\langle\gamma^\times\rangle\varphi)^{a,x,y}, (\omega)^b]^y}{\Phi, (\langle\gamma^\times\rangle\varphi)^{a,x}, (\psi \wedge \omega)^b}\,\wedge\\ \vdots\\ \dfrac{\Phi, (\varphi \wedge \langle\gamma\rangle\langle\gamma^\times\rangle\varphi)^{a,x,y}, (\omega)^b}{\Phi, (\langle\gamma^\times\rangle\varphi)^{a,x}, (\omega)^b}\,\mathrm{clo}_y\end{array}}{\Phi, (\langle\gamma^\times\rangle\varphi)^{a,x}, (\psi \wedge \omega)^b}\,\wedge\\ \vdots\\ \dfrac{\Phi, (\varphi \wedge \langle\gamma\rangle\langle\gamma^\times\rangle\varphi)^{a,x}, (\psi)^b}{\Phi, (\langle\gamma^\times\rangle\varphi)^{a}, (\psi)^b}\,\mathrm{clo}_x\end{array} \qquad \begin{array}{c}\dfrac{\begin{array}{c}\dfrac{[\Phi, (\langle\gamma^\times\rangle\varphi)^{a,x,y}, (\varphi)^b]^y \qquad [\Phi, (\langle\gamma^\times\rangle\varphi)^{a,x}, (\omega)^b]^x}{\Phi^\varepsilon, (\langle\gamma^\times\rangle\varphi)^{a,x}, (\psi \wedge \omega)^b}\,\wedge\\ \vdots\\ \dfrac{\Phi, (\varphi \wedge \langle\gamma\rangle\langle\gamma^\times\rangle\varphi)^{a,x,y}, (\varphi)^b}{\Phi, (\langle\gamma^\times\rangle\varphi)^{a,x}, (\varphi)^b}\,\mathrm{clo}_y\end{array} \qquad [\Phi, (\langle\gamma^\times\rangle\varphi)^{a,x}, (\omega)^b]^x}{\Phi, (\langle\gamma^\times\rangle\varphi)^{a,x}, (\psi \wedge \omega)^b}\,\wedge\\ \vdots\\ \dfrac{\Phi, (\varphi \wedge \langle\gamma\rangle\langle\gamma^\times\rangle\varphi)^{a,x}, (\psi)^b}{\Phi, (\langle\gamma^\times\rangle\varphi)^{a}, (\psi)^b}\,\mathrm{clo}_x\end{array}}{\Phi, (\langle\gamma^\times\rangle\varphi)^{a}, (\psi \wedge \omega)^b}\,\wedge$$

□

5.3 Strategies

Based on the interchangeability of simple rules, we can devise a new type of strategy:

Definition 5.2 (Saturation Search). *We first try to saturate the current sequent as much as possible, i.e., whenever we can apply one of the simple rules, we do so, until either no simple rule can be applied, or a cycle is detected. Now that the sequent is saturated, we try to apply the* mod_m *rule to two of the formulae in the sequent. Before the actual* mod_m *rule application, we use* weak *rule application in order to reach the sequent containing only the two formulae to which the* mod_m *rule is applied. If no proof is found, we backtrack to before the aforementioned* weak *rule applications and see if we can apply the* mod_m *rule to a different pair of formulae in the sequent. If no more pair of formulae is left, no proof could be found regardless.*

There are a few variations of this saturation search. Firstly, with regards to the clo rule. We can try to apply the closure rule as late as possible, or as early as possible.

Trying to apply the clo rule as late as possible should be safe to do, as this corresponds to moving the clo rule application up, which we can do according to lemma 5.1. But the sequent to which this clo rule will be applied, would be almost completely saturated (save for the formulae with a × as the outermost operator). And to match an assumption further up in the proof and discharge the sequent, we would need to retain all the saturated side formulae, or regenerate them after a mod_m rule application.

However, we can also apply the clo rule as early as possible, which might not be complete (i.e. able to find a proof for any valid sequent), but the resulting proofs will usually have fewer side formulae to match in the assumption, thus if a proof is found, the proof will often be shorter, and found more easily.

We can also try to apply the mod_m rule as early as possible, but if it fails, we backtrack to before the mod_m application and try the next rule application. This should result in shorter proofs, since sequents can be weakened before they are fully saturated.

This leaves us with 4 distinct proof strategies, which we can give distinct names, so we can refer to them later in the thesis:

Definition 5.3 (The Procrastination Strategy). *The strategy in which we apply both the* clo *and the* mod_m *rule as late as possible.*

This ensures the sequent is saturated before applying the mod_m rule, and the closure assumptions to be matched are also saturated.

Definition 5.4 (The Pre-Emptive mod_m Strategy)**.** *The strategy in which we apply the* clo *as late as possible, but the* mod_m *as early as possible.*

This prevents potential unnecessary further fixpoint unfoldings.

Definition 5.5 (The Simple clo Strategy)**.** *The strategy in which we apply the* clo *as early as possible, but the* mod_m *rule as late as possible.*

This should allow for simpler assumption to match, but might not be complete.

Definition 5.6 (The Greedy Strategy)**.** *The strategy in which we apply both the* clo *and the* mod_m *rule as early as possible.*

This should prevent unnecessary fixpoint unfoldings, and make for simpler closure assumptions to match, but might not be complete.

Lastly, we assume that we can leave out the × rule, as it does the exact same transformation as the clo rule, but allows for some cycles that are hard to detect in our cycle detection, where via a combination of × and clo rule applications, we end up with a sequent $\Phi, \varphi^a, \varphi^b$ somewhere above Φ, φ^a (an example is discussed in section 7.2.11).

There might be proofs that require the application of the × rule instead of a clo rule, because the clo rule would prevent certain unfoldings of fixpoint formulae higher up in the proof, which might be necessary in finding a proof. However, we could find no examples of proofs where this is necessary, and we conjecture that even without trying to apply the × rule, the proof system is still complete. Proving this falls outside of the scope of this project.

6 Implementation

6.1 Development Tools and Technology

6.1.1 Rascal

In order to apply Automated Theorem Proving, we need a way to parse a sequent that needs to be proven, a way to represent both the sequent and the resulting proof, and an algorithm that can search for such a proof, and backtrack to a previous sequent if a proof could not be found. In order to apply each individual rule, pattern matching will have to be applied.

All of these features are provided by the meta-programming language of Rascal [11]. Rascal is built for metaprogramming, and can be used for domain-specific languages. One could consider the CloG proof system to be one such domain-specific language. It has its own grammar, syntax, and rules of operation. Rascal provides a set of primitives that can easily be used to represent Game Logic formulae, and CloG proofs. Moreover, Rascal provides many parsing and pattern matching capabilities, which makes it easy to process inputted logic formulae, and represent it internally in a data structure.

Working in Rascal also means being able to reuse some of the code of Worthington's proof transformation tool [22]. This tool provides a grammar for CloG proofs, and an abstract syntax tree to represent them. Moreover, it provides a way to display these proofs in LaTeX, which will be useful for this thesis.

One concern with Rascal is that it might be somewhat slow and unoptimized in its proof search, but as this project is more of a proof of concept / prototype, speed is not one of the main concerns.

6.1.2 Eclipse

The Rascal documentation recommends working in the Eclipse IDE and using an Eclipse plugin for Rascal, which offers support for Rascal projects and syntax highlighting.

6.1.3 Git & Github

For version control, we simply use Git and Github, as these are industry standards, and we do not need anything more. The public Git repository for this project can be found at `https://github.com/HanMeerholz/CloG-ATP`.

6.2 Pseudocode

What follows is the pseudocode for the various parts of the tool.

Next to the standard **if-elsif-else** statements, **for** and **for all** loops, **return**, and **break** statements, to prevent many nestings of if-statements, a **continue** keyword is included that immediately progresses to the next iteration of the innermost loop, or of another loop if otherwise specified. Moreover, some custom data types, either defined in [22], or in the Rascal implementation of this tool itself, are used. Please note that the terminology in [22], and subsequently in the pseudocode and Rascal implementation of this tool, to denote an annotated formula is "term", not to be confused with how "term" is defined in [7] (as either a full game logic formula, or a game within a game logic formula).

As parts of this code are implementation-dependent, this might not fully be considered pseudocode.

6.2.1 Data Types

A CloG sequent is proven, if we can derive a CloG proof tree for it. A CloG proof tree is either an empty leaf, a discharge of a closure rule, or the application of a rule and the subproofs for its resulting sequent(s). To represent a CloG proof tree node, we use a data type that can be of the following forms:

- a `CloGLeaf`
- a `disClo`: a discharge of the sequent associated with the closure rule application, which contains the sequent to which it is applied, and the name of the fixpoint formula associated with the closure rule application
- a `CloGUnaryInf`: the application of a rule with a single premise, which contains the rule applied and the proof for its premise
- a `CloGBinaryInf`: the application of a rule with two premises (i.e. the $\wedge$ rule), which contains the proofs of both of its premises

Next to the data type for a proof tree, we also use the following data types:

- the `CloGName` data type is a (possibly subscripted) name for a formula in CloG
- the `GameLog` data type represents a game logic formula
- the `Game` data type represents a game formula, whatever can be in a game logic modality
- a `CloGTerm` represents an annotated formula, and consists of a `GameLog` and a list of `CloGName`s
- a `CloGSequent` represents a CloG sequent and is a list of `CloGTerms`
- a `CloSeqs` is a map, mapping from the name of a fixpoint formula to a tuple containing the `CloGSequent` which was the premise for the associated closure rule application, and an index indicating which term in this sequent contains the relevant fixpoint formula. Thus the `CloSeqs` data type is defined as $map[CloGName\ name,\ tuple[CloGSequent\ context,\ int\ termIdx]\ fpSeq]$. If $closeqs$ is of this type, referring to $cloSeqs[x]$ will return the $\langle CloGSequent,\ int\rangle$ pair associated with the name x.

6.2.2 Proof Search Strategies

Algorithms 1, 2, 3, and 4 describe depth-first proof searches, that use the saturation strategy, described in section 5.2.2. There are 4 variants corresponding to rule orderings described in section 5.3.

Algorithm 1 corresponds to the Procrastination Strategy (definition 5.3), and tries to apply both the clo rule and the mod_m rule late.

Algorithm 2 corresponds to the Pre-Emptive mod_m Strategy (definition 5.4) and tries to apply the mod_m rule early, and the clo rule late

Algorithm 3 corresponds to the simple clo Strategy (definition 5.5), and tries to apply the clo early and the mod_m rule late.

Algorithm 1 Proof Search (Procrastination Strategy)

```
procedure proofSearch($CloGSequent\ seq,\ CloSeqs\ cloSeqs,\ List[CloGSequent]\ fpSeqs$)
    if $proof \leftarrow tryDisClo(seq,\ cloSeqs) \neq noProof()$ then
        return $proof$
    end if
    if $detectCycles(seq,\ fpSeqs)$ then
        return $cantApply()$    ▷ if there is a cycle, we cannot apply the rule that was applied one level of
                                  recursion back, and we must backtrack
    end if
    if $proof \leftarrow tryApplyAx1(seq) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyChoice(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$    ▷ if the rule can't be applied, move to the next one; if it can be applied
                            either return the resulting proof or return that no proof could be found
    else if $proof \leftarrow tryApplyDChoice(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyConcat(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyTest(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyDTest(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyOr(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyAnd(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyIter(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyClo(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyModm(seq,\ cloSeqs,\ fpSeqs) \neq cantApply() \wedge$
  $tryApplyModm(seq,\ cloSeqs,\ fpSeqs) \neq noProof()$ then
        return $proof$    ▷ only return the proof if a complete proof is found; if the rule could be
                            applied, but no proof could be found, also move on to the next rule
    end if
    return $noProof()$    ▷ if none of the rules could be successfully applied, no proof is returned
end procedure
```

Algorithm 2 Proof Search (Pre-Emptive modm Strategy)

```
procedure PROOFSEARCH($CloGSequent\ seq,\ CloSeqs\ cloSeqs,\ List[CloGSequent]\ fpSeqs$)
    if $proof \leftarrow tryDisClo(seq,\ cloSeqs) \neq noProof()$ then
        return $proof$
    end if
    if $detectCycles(seq,\ fpSeqs)$ then
        return $cantApply()$    ▷ if there is a cycle, we cannot apply the current rule, and we must backtrack
    end if
    if $proof \leftarrow tryApplyAx1(seq) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyModm(seq,\ cloSeqs,\ fpSeqs) \neq cantApply() \wedge tryApplyModm(seq,\ cloSeqs,\ fpSeqs) \neq noProof()$ then
        return $proof$    ▷ only return the proof if a complete proof is found; if the rule could be
                          applied, but no proof could be found, also move on to the next rule
    else if $proof \leftarrow tryApplyChoice(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$    ▷ if the rule can't be applied, move to the next one; if it can be applied
                          either return the resulting proof or return that no proof could be found
    else if $proof \leftarrow tryApplyDChoice(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyConcat(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyTest(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyDTest(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyOr(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyAnd(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyIter(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyClo(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    end if
    return $noProof()$    ▷ if none of the rules could be successfully applied, no proof is returned
end procedure
```

Algorithm 3 Proof Search (Simple Clo Strategy)

```
procedure PROOFSEARCH($CloGSequent\ seq,\ CloSeqs\ cloSeqs,\ List[CloGSequent]\ fpSeqs$)
    if $proof \leftarrow tryDisClo(seq,\ cloSeqs) \neq noProof()$ then
        return $proof$
    end if
    if $detectCycles(seq,\ fpSeqs)$ then
        return $cantApply()$   ▷ if there is a cycle, we cannot apply the current rule, and we must backtrack
    end if
    if $proof \leftarrow tryApplyAx1(seq) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyClo(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyChoice(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$   ▷ if the rule can't be applied, move to the next one; if it can be applied either return the resulting proof or return that no proof could be found
    else if $proof \leftarrow tryApplyDChoice(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyConcat(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyTest(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyDTest(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyOr(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyAnd(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyIter(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyModm(seq,\ cloSeqs,\ fpSeqs) \neq cantApply() \wedge tryApplyModm(seq,\ cloSeqs,\ fpSeqs) \neq noProof()$ then
        return $proof$   ▷ only return the proof if a complete proof is found; if the rule could be applied, but no proof could be found, also move on to the next rule
    end if
    return $noProof()$   ▷ if none of the rules could be successfully applied, no proof is returned
end procedure
```

Algorithm 4 Proof Search (Greedy Strategy)

```
procedure PROOFSEARCH($CloGSequent\ seq,\ CloSeqs\ cloSeqs,\ List[CloGSequent]\ fpSeqs$)
    if $proof \leftarrow tryDisClo(seq,\ cloSeqs) \neq noProof()$ then
        return $proof$
    end if
    if $detectCycles(seq,\ fpSeqs)$ then
        return $cantApply()$    ▷ if there is a cycle, we cannot apply the current rule, and we must backtrack
    end if
    if $proof \leftarrow tryApplyAx1(seq) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyModm(seq,\ cloSeqs,\ fpSeqs) \neq cantApply() \wedge$
            $tryApplyModm(seq,\ cloSeqs,\ fpSeqs) \neq noProof()$ then
        return $proof$    ▷ only return the proof if a complete proof is found; if the rule could be
                          applied, but no proof could be found, also move on to the next rule
    else if $proof \leftarrow tryApplyClo(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyChoice(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$    ▷ if the rule can't be applied, move to the next one; if it can be applied
                          either return the resulting proof or return that no proof could be found
    else if $proof \leftarrow tryApplyDChoice(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyConcat(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyTest(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyDTest(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyOr(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyAnd(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    else if $proof \leftarrow tryApplyIter(seq,\ cloSeqs,\ fpSeqs) \neq cantApply()$ then
        return $proof$
    end if
    return $noProof()$    ▷ if none of the rules could be successfully applied, no proof is returned
end procedure
```

Finally, algorithm 4 corresponds to the Greedy Strategy (definition 5.6) and tries to apply both rules as early as possible.

All of these proof search algorithms take as input the `CloGSequent` that needs to be proven, a `CloSeqs`, which is initially empty, and a list of `CloGSequents` for cycle detection, also initially empty. If a proof is found, the proof is returned by the algorithm as a `CloGProof`, otherwise `noProof()` is returned, indicating no proof was found, or `cantApply()` is returned, in case of a cycle detection. The distinction between these two return values will be explained later, for the rule application algorithms.

First, each of the variants of the algorithm call the $tryDisClo()$ function to check whether the current branch can be closed by finding the assumption for an earlier applied clo rule. After this, to prevent infinite recursion, a check is done whether there are any other cycles present, and `cantApply()` is returned if this is the case. Finally, it tries applying each of the rules, and if any of them find a proof, that proof is returned. If none of the rules applied find a proof, `noProof()` is returned. As for the other rule applications besides mod_m and clo in the pseudocode, we first try the $\mathsf{Ax1}$ axiom, since, if successful, $\mathsf{Ax1}$ closes the branch. Then we try all the local rules which reduce the formulae in the sequent, $\sqcup$, $\sqcap$, ;, ?, !, $\vee$, and $\wedge$, and finally, we try the $*$ rule, which unfolds fixpoint formulae.

This pseudocode refers to functions corresponding to each of the CloG rules, named $tryApply\langle ruleName\rangle()$. These functions all take the same parameters as the main $proofSearch()$ algorithm, namely the `CloGSequent` to be proven, and the map of closure sequents. The exception is the $tryApplyAx1()$ rule, which does not have a cyclic dependency with the $proofSearch()$ algorithm, and only needs the current sequent as an input. Each of the remaining functions tries to apply their rule in some way (on a specific term for example), and then calls $proofSearch()$ on the resulting sequent from applying the rule. If the rule can not be applied, `cantApply()` is returned; if the rule can be applied, but no proof can be found for the resulting sequent, `noProof()` is returned. This distinction is made to account for the exchangeability of rule applications described in section 5.2.2. If `cantApply()` is returned, the main $proofSearch()$ algorithm tries to apply the next rule. However, if `noProof()` is returned, the main $proofSearch()$ algorithm itself also returns `noProof()` and no backtracking is done.

There is one exception to this. If a $tryApplyModm()$ function returns `noProof()`, the other rules are still tried (as long as there are other rules after the mod_m rule, thus in algorithms 1 and 3). This is because all the other rules can be applied in any order to get the same saturated sequent, but as discussed in section 5.2.2, this is not the case for the mod_m rule.

$proofSearch()$ returns `cantApply()` rather than `noProof()` upon detecting a cycle. This has to do with how a cycle detection should be treated. For all four of the strategies discussed in section 5.3, most of the rule applications are exchangeable, which means if one of the $tryApply\langle Rule\rangle()$ functions returns a $noProof()$, we know we should not try to apply the same rules in a different order, thus we backtrack all the way until we come to a point where we applied a non-simple rule, i.e. the mod_m or clo rule, or to the beginning of the proof search, and continue our search from there. However, if we detect an undesired cycle, we only want to backtrack to the previous sequent, and we still want to try to apply the other rules from that point on. To backtrack only one step like this, we can simply return $cantApply()$ rather than $noProof()$.

From all this, we can say that this is not an exhaustive depth-first search algorithm, as that is unnecessary. Not all rule application orders are considered. The only backtracking (more than a single step when a rule cannot be applied or a cycle is detected) is done upon a failed rule application (that is not the mod_m rule).

6.2.3 Closure Sequent Discharge

The pseudocode for the $tryDisClo()$ can be found in algorithm 5. It checks whether the current sequent can reach any of the saved sequents in the `CloSeqs` by just applying weak or exp rules, and if so, it returns a subproof ending in a `disClo()`.

6.2.4 Cycle Detection

The $detectCycles()$ algorithm (algorithm 6) goes through all the saved fixpoint sequents (for all the applied $*$ and $\times$ rules so far in the current branch), and if the current sequent matches one of these fixpoint sequents, it means an undesired cycle is detected. The only cycles that are desired are those that can be used to discharge the sequent for a closure rule, and we already checked for that in $tryDisClo()$.

Algorithm 5 Closure Discharge

```
procedure tryDisClo($Sequent\ seq,\ CloSeqs\ cloSeqs$)
    for all $CloGName\ n$ in $cloSeqs$ do                    ▷ if sequents can be discharged, do so
        $fpTerm \leftarrow cloSeqs[n].context[cloSeqs[n].termIdx]$
        for $int\ termIdx \leftarrow 0,\ seq.length - 1$ do
            if $seq[termIdx]$ is of the form "$((\langle\gamma^\times\rangle\varphi)^a$" and $fpTerm$ is of the form "$((\langle\gamma^\times\rangle\varphi)^b$" and $b{+}n \subseteq a$ then
                $resSeq \leftarrow cloSeqs[name].context - fpTerm + (\langle\gamma^\times\rangle\varphi)^{b+n}$
                $proof \leftarrow proofSearchWeakExp(seq,\ resSeq)$
                replace the leaf of $proof$ by $disClo(resSeq,\ name)$
                return $proof$
            end if
        end for
    end for
    return $noProof()$
end procedure
```

Algorithm 6 Cycle Detection

```
procedure detectCycles($Sequent\ seq,\ List[CloGSequent]\ fpSeqs$)
    for all $fpSeq$ in $fpSeqs$ do
        if $seq == fpSeq$ then
            return true
        end if
    end for
    return false
end procedure
```

6.2.5 Rule Applications

The Ax1 axiom and mod_m rule are special, since they require exactly 2 terms in their premise. Thus, to apply them, we see if we can reach these 2 terms by weak or exp rules only, and only then we apply the Ax1 axiom or mod_m rule. The pseudocode for the $tryApplyAx1()$ and $tryApplyModm()$ functions is provided in algorithms 7 and 8, respectively.

The remaining rules can be applied to any one of the terms of the input sequent, thus we loop over these terms and try to apply the rule to each of them. Algorithm 9 describes the $tryApplyOr()$ function, which tries to apply the $\vee$ rule to each of the formulae in the sequent, and if successful, calls the main $proofSearch()$ function on the resulting sequent. If the rule can be applied to a sequent, but no subproof(s) can be found, we return `noProof()` or `cantApply()` immediately before checking the rest of the formulae in the sequent, because once again, the order does not matter.

The pseudocode for the $tryApplyChoice()$, $tryApplyDChoice()$, $tryApplyConcat()$, $tryApplyTest()$, and $tryApplyDTest()$ functions is not provided, but these functions work analogously to the $tryApplyOr()$ function, but with the corresponding CloG rule applied instead of the $\vee$ rule.

When the $\wedge$ rule is applied, it results in 2 sequents. In $tryApplyAnd()$ (see algorithm 10), $proofSearch()$ is called for both of these sequents.

The $tryApplyIter()$ function is similar to the $tryApplyOr()$ function, but the $*$ rule has a side condition, such that for each term, the fixpoint ordering must hold. For this, the `CloSeqs` map will have to be checked. If the rule can be applied, the full sequent is saved in $fpSeqs$. The pseudocode for the $tryApplyIter()$ function can be found in algorithm 11.

The pseudocode for the $tryApplyClo()$ function can be found in algorithm 12. This function needs to account for the same side condition as the $tryApplyIter()$ function. Moreover, the $tryApplyClo()$ function adds a closure sequent to the `CloSeqs` map. The $tryApplyClo()$ function also does not immediately return `noProof()` or `cantApply()` if the rule can be applied to a sequent, but no subproof(s) can be found, like the other rules. This is because we have not been able to prove that we can exchange two consecutive clo rule

Algorithm 7 ax1 Rule Application

```
procedure TryApplyAx1($CloGSequent\ seq$)
    for $int\ termIdx \leftarrow 0,\ seq.length - 1$ do
        if $seq[termIdx]$ is of the form “$p^a$” then
            for $int\ termIdx2 \leftarrow 0,\ seq.length - 1$ do
                if $seq[termIdx2]$ is of the form “$(\neg p)^b$” then
                    $weakenTo \leftarrow [p^\epsilon, (\neg p)^\epsilon]$
                    $proof \leftarrow proofSearchWeakExp(seq,\ weakenTo)$
                    replace the leaf of $proof$ by $CloGUnaryInf(resSeq,\ ax1(),\ CloGleaf())$
                    return $proof$
                end if
            end for
        end if
    end for
    return $cantApply()$
end procedure
```

Algorithm 8 modm Rule Application 2

```
procedure TryApplyModm($CloGSequent\ seq,\ CloSeqs\ cloSeqs,\ List[CloGSequent]\ fpSeqs$)
    for $int\ termIdx \leftarrow 0,\ seq.length - 1$ do
        if $seq[termIdx]$ is of the form “$(\langle g \rangle \varphi)^a$” then
            for $int\ termIdx2 \leftarrow 0,\ seq.length - 1$ do
                if $seq[termIdx2]$ is of the form “$(\langle g^d \rangle \psi)^b$” then
                    $weakenTo \leftarrow [(\langle g \rangle \varphi)^a, (\langle g^d \rangle \psi)^b]$
                    $resSeq \leftarrow [\varphi^a, \psi^b]$
                    $remProof \leftarrow proofSearch(resSeq,\ cloSeqs,\ fpSeqs)$
                    if $remProof \neq noProof() \vee cantApply()$ then
                        $proof \leftarrow proofSearchWeakExp(seq,\ weakenTo)$
                        replace the leaf of $proof$ by $CloGUnaryInf(resSeq,\ modm(),\ proof)$
                        return $proof$
                    end if
                end if
            end for
        end if
    end for
    return $cantApply()$
end procedure
```

Algorithm 9 or Rule Application

```
procedure tryApplyOr($CloGSequent\ seq,\ CloSeqs\ cloSeqs,\ List[CloGSequent]\ fpSeqs$)
    for $termIdx \leftarrow 0,\ seq.length - 1$ do
        if $seq[termIdx]$ is of the form “$(\varphi \vee \psi)^a$” then
            $subSeq \leftarrow seq$ where $seq[termIdx]$ is replaced by two terms $[\varphi^a,\ \psi^a]$   ▷ the other terms shift an index
            $proof \leftarrow proofSearch(subSeq,\ cloSeqs,\ fpSeqs)$
            if $proof \neq noProof() \vee cantApply()$ then
                return $ClogUnaryInf(seq,\ orR(),\ proof)$
            end if
            return $proof$                              ▷ return either noProof() or cantApply()
        end if
    end for
    return $cantApply()$
end procedure
```

Algorithm 10 and Rule Application

```
procedure tryApplyAnd($CloGSequent\ seq$, $CloSeqs\ cloSeqs$, $List[CloGSequent]\ fpSeqs$)
    for $termIdx \leftarrow 0,\ seq.length - 1$ do    ▷ try applying the rule to each of the formulae in the sequent
        if $seq[termIdx]$ is of the form "$(\varphi \wedge \psi)^a$" then    ▷ a list of two sequents, both containing the same formulae as the premise, except for the $n$th formula, which was replaced
            $subSeq1 \leftarrow seq$ where $seq[termIdx]$ is replaced by $\varphi^a$
            $proofL \leftarrow proofSearch(subSeq1,\ cloSeqs,\ fpSeqs)$
            if $proofL = noProof() \vee cantApply()$ then
                return $proofL$    ▷ rule application ordering should not matter; if the rule can be applied, but no recursive proof is found, there is no proof
            end if
            $subSeq2 \leftarrow seq$ where $seq[termIdx]$ is replaced by $\psi^a$
            $proofR \leftarrow proofSearch(subSeq2,\ cloSeqs,\ fpSeqs)$
            if $proofR = noProof() \vee cantApply()$ then
                return $proofR$
            end if
            return $CloGBinaryInf(seq,\ proofL,\ proofR)$
        end if
    end for
    return $cantApply()$
end procedure
```

Algorithm 11 iter Rule Application

```
procedure tryApplyIter($CloGSequent\ seq$, $CloSeqs\ cloSeqs$, $List[CloGSequent]\ fpSeqs$)
    for $termIdx \leftarrow 0,\ seq.length - 1$ do    ▷ to apply the iter rules, we need access to the list of closure sequents
        if $seq[termIdx]$ is of the form "$(\langle\gamma^*\rangle\varphi)^a$" then
            for all names $x$ in $a$ do
                $savedFormula \leftarrow$ the fixpoint formula in $cloSeqs$ for the name $x$
                if not $fpLessThanOrEqualTo(savedFormula, \langle\gamma^*\rangle\varphi)$ then
                    continue to next $termIdx$    ▷ if the side condition fails, the application fails
                end if
            end for
            $subSeq \leftarrow seq$ where $seq[termIdx]$ is replaced by $(\varphi \vee \langle\gamma\rangle\langle\gamma^*\rangle\varphi)^a$
            add $seq$ to $fpSeqs$    ▷ save the current sequent in fpSeqs to prevent cycles
            $proof \leftarrow proofSearch(subSeq,\ cloSeqs,\ fpSeqs)$
            if $proof \neq noProof() \vee cantApply()$ then
                return $ClogUnaryInf(seq,\ iterR(),\ proof)$
            end if
            return $proof$
        end if
    end for
    return $cantApply()$
end procedure
```

applications in section 5.2.2, thus we need to try the `clo` rule for each of the terms in the sequent, even if a subproof returns a `noProof()` or `cantApply()`.

Algorithm 12 clo Rule Application

```
procedure tryApplyClo($CloGSequent\ seq,\ CloSeqs\ cloSeqs,\ List[CloGSequent]\ fpSeqs$)
    for $termIdx \leftarrow 0,\ seq.length - 1$ do
        if $seq[termIdx]$ is of the form "$(\langle\gamma^{\times}\rangle\varphi)^{a}$" then
            for all names $x$ in $a$ do
                $savedFormula \leftarrow$ the fixpoint formula in $cloSeqs$ for the name $x$
                if not $fpLessThanOrEqualTo(savedFormula, \langle\gamma^{\times}\rangle\varphi)$ then
                    continue to next $termIdx$
                end if
            end for
            add $seq$ to $fpSeqs$
            add $(x_{cloSeqs.length},\ \langle seq,\ termIdx\rangle)$ to $cloSeqs$    ▷ connect the fixpoint formula to a new name subscripted the length of the current $cloSeqs$
            $newSeq \leftarrow seq$ where $seq[termIdx]$ is replaced by $(\varphi \wedge \langle\gamma\rangle\langle\gamma^{\times}\rangle\varphi)^{ax}$
            $proof \leftarrow proofSearch(newSeq,\ cloSeqs,\ fpSeqs)$
            if $proof \neq noProof() \vee cantApply()$ then
                return $CloGUnaryInf(subSeq,\ clo(),\ remProof)$
            end if
        end if
    end for
    return $noProof()$
end procedure
```

In the rule application functions, often a condition like "**if** *seq* is of the form "⟨formula⟩" **then** ..." is present. We will not go into detail about how exactly the algorithm checks for this formula form, because this is implementation specific. Pattern matching is one of Rascal's biggest strengths, thus this is quite simple in the actual implementation.

6.2.6 Helper Functions

The *weak* and *exp* rule applications are not included, as those are handled by the $proofSearchWeakExp()$ algorithm (algorithm 13). This algorithm takes two `CloGSequents` as input. The algorithm tries to apply weakening and expanding rules to the first sequent, such that it results in the second sequent. It recursively calls itself until either the first sequent is empty, in which case no proof could be found, or the first sequent is equal to the second, in which case it returns an incomplete proof, with the final rule application being a "dummy" weakening rule to a $CloGLeaf()$. This leaf will have to be replaced by the appropriate subproof, as is done in $tryDisClo()$, $tryApplyAx1()$, and $tryApplyModm()$.

The pseudocode for $tryApplyIter()$ and $tryApplyClo()$ also refers to the helper function $fpLessThenOrEqualTo()$, which is related to the fixpoint ordering defined in [7], and pseudocode for it is supplied in algorithm 14. This function takes two `GameLog` formulae as input, and outputs a simple `true` or `false` on whether its first parameter is smaller than or equal to its second in fixpoint formula ordering. $fpLessThenOrEqualTo()$ itself refers to a $subTerm()$ helper function (see algorithm 15), which takes `Games` as input, and returns whether its first parameter is a subterm of its second parameter.

6.2.7 Post-Processing

Technically, in order for a `clo` rule application to be valid in a proof, there needs to be an assumption somewhere above the prove node where the `clo` rule was applied. Algorithms 1, 2, 3, and 4 are not strict enough and can produce proofs that use `clo` rule applications without an assumption.

This is not a difficult issue to resolve however, since we can replace these redundant `clo` rule applications simply by × applications. The × rule transforms the sequent in the same way as the `clo` rule does, except that

Algorithm 13 weak/exp Proof Search

```
procedure proofSearchWeakExp($Sequent\ seqFrom,\ Sequent\ seqTo$)
    if $seqFrom$ is empty then
        return $noProof()$
    end if
    if $seqFrom = seqTo$ then
        return $CloGUnaryInf(seqFrom, weak(), CloGLeaf())$    ▷ We return an incomplete proof here; it is up to the parent function to make sure this proof is properly closed
    end if
    for $i \leftarrow 0,\ seqFrom.length - 1$ do
        for $j \leftarrow 0,\ seqTo.length - 1$ do
            if $seqFrom[i].formula = seqTo[j].formula$ then
                for all $CloGName\ n$ in $seqFrom[i].label - seqTo[j].label$ do
                    $newSeq = seq1 - seq1[i] + term(seq1[i].formula,\ seq[i].label - n)$
                    $proof = proofSearchWeakExp(newSeq,\ seqTo,\ cloSeqs,\ leaf)$
                    if $proof \neq noProof()$ then
                        return $CloGUnaryInf(seq1,\ exp(),\ proof)$
                    end if
                end for
            end if
        end for
        $newSeq = seq1 - seq1[i]$
        $proof = proofSearchWeakExp(newSeq,\ seqTo)$
        return $CloGUnaryInf(seq1,\ weak(),\ proof)$
    end for
end procedure
```

Algorithm 14 Less Than Or Equal To (Fixpoint Formulae)

```
procedure fpLessThanOrEqualTo($GameLog\ phi,\ GameLog\ psi$)
    if $phi$ is of the form $\langle fp0\rangle\varphi$ and $psi$ is of the form $\langle fp1\rangle\psi$, where $fp0$ is of the form $\gamma^*$ or $\gamma^\times$, and $fp1$ is of the form $\delta^*$ or $\delta^\times$ then
        return $subTerm(fp1,\ fp0)$
    end if
    return false
end procedure
```

Algorithm 15 Subterm Of

```
procedure subTerm($Game\ g,\ Game\ h$)
    if $g$ is a subterm of $h$, i.e., the syntax tree of $g$ is a subtree of the syntax tree of $h$ then
        return true    ▷ simple pattern matching
    end if
    return false
end procedure
```

it does not add a name to the formula's annotation. Since there is no assumption to be discharged anymore, this name is unnecessary. And since an extra name in the annotations only makes the side condition for consequent $*$, $\times$, and clo rules stricter, dropping it will not invalidate any of the rule applications above the proof node where clo is replaced by $\times$.

When replacing a clo rule application by a $\times$ rule application, we need to not only update the one rule application, but we need to drop the name corresponding to the original clo rule application from each sequent above the proof node. Pseudocode for this procedure is provided in algorithm 16. It takes a `CloGProof` as an input and also outputs a `CloGProof`.

Algorithm 16 Replace clo Rule Application by $\times$ Rule Application

```
procedure replaceUnusedClos($CloGProof$)
    for all $CloGUnaryInf()$ nodes $node$ with a $clo$ rule application (with associated $CloGName$ $n$) do
        if there no $disClo$ appears above $node$, discharging the assumption associated with name $n$ then
            replace the rule for $node$ by $\times$
            for all sequents $seq$ above $node$ do
                for all $CloGTerm$ $term$ in $seq$ do
                    if $n$ appears in the label at $term$ then
                        remove $n$ from the label
                    end if
                end for
            end for
        end if
    end for
end procedure
```

6.3 Rascal Code Organisation

All of the Rascal code relevant for this project is provided in appendix C.

The Rascal code for this project is split up into two submodules for the program itself, and two submodules for testing.

6.3.1 CloG_Base

The `CloG_Base` submodule (appendix C.1) defines the concrete syntax for inputted CloG proofs and sequents, the *abstract syntax tree* (AST) for a CloG proof, and a way to transform these ASTs into a proof tree represented in LaTeX. The files in this submodule are the same as in [22], but have had some small updates.

`CloGSyntax.rsc` (listing C.1.1) is a simplified version of the original: the concrete syntax merely needs to account for CloG sequents, rather than full proofs. Apart from this simplification, the only change made to the file is how `Id` is defined, since the subscripted input was not working properly.

`GLASTs.rsc` (listing C.1.3) defines the abstract syntax tree for a CloG proof. Again, little changes were made with respect to the code in [22]. A definition (alias) of `CloGSequent` was added, and each `CloGTerm` now has an `active` parameter, which indicates whether a rule is applied to this term (annotated formula).

`CST2AST_CloG.rsc` (listing C.1.2) has also been updated accordingly to work with sequents rather than full proofs. The `active` parameter for each term in the sequent is set to `false` by default.

Finally, `LaTeXOutput.rsc` has been updated to display the active terms in red.

6.3.2 ATP

The `ATP` submodule (appendix C.2) deals with the automated theorem prover itself. It defines the relevant data types for the recursive search, defines the main proof search algorithm, the algorithms for each of the rule applications, and helper functions. Moreover, it contains the main tool for user interaction.

`ATP_Base.rsc` (listing C.2.1) defines the data types used in the proof search, such as the `CloSeqs` data type as explained in section 6.2.1, a `MaybeProof` which acts as the return type for the proof search algorithm, and is either a `CloGProof`, a `noProof()` or a `cantApply()`, a `MaybeSequent` that either returns a `CloGSequent`, or a `noSequent()`, and a `MaybeSequents` that either returns a list of `CloGSequents`, or a `noSequents()`. Finally, the helper functions for fixpoint ordering, see algorithm 14 and 15, are provided.

`ProofSearch.rsc` (listing C.2.2) contains the main proof search algorithm in the `proofSearch()` function. The implementation of the algorithm is like the ones in the pseudocode (algorithms 1, 2, 3, and 4 in section 6.2.2; the exact rule ordering is changed manually in the code), with an addition of a maximum depth (see appendix A.2), and a line to remove duplicates (see appendix A.1). `ProofSearch.rsc` also contains a cycle detection function `detectCycles()` (algorithm 6), a function `tryDisClo()` that tries closing a branch by discharging a closure sequent (algorithm 5), and the `proofSearchWeakExp()` function that tries finding a (incomplete) proof from one sequent to another by only applying the weak and exp rules (algorithm 13).

`RuleApplications.rsc` (listing C.2.3) contains the functions corresponding to algorithms 7, 8, 9, 10, 11, and 12, and the functions corresponding to all the remaining CloG rules. Some minor changes to the pseudocode are discussed in appendix A.3.

`PostProcess.rsc` (listing C.2.4) handles the post-processing phase of the automated theorem prover, which consists only of a function `replaceUnusedClos()` (see algorithm 16), which replaces each clo rule application that does not have a discharged assumption by a × rule application instead. This function also has a helper function `replaceCloAt()`, which is called for each clo rule application that needs to be replaced.

Finally, `CloG_ATP_Tool.rsc` (listing C.2.5) is a module based on the `GLTool.rsc` file in [22]. This module provides the main interface that a user would interact with. It provides functions that take as input a file name of a `.seq` file (containing sequents with the syntax defined in `CloGSyntax.rsc`), create an AST for the sequent in said file, call the `proofSearch()` function on that AST, do the necessary post-processing, and output the resulting `CloGProof` to a LaTeX prooftree format (or log "fail!" if no proof could be found).

6.3.3 unitTests & integrationTests

The `unitTests` and `integrationTests` submodules are for testing and will be further described in section 7.

7 Testing & Analysis

7.1 Unit Tests

Each function that did not have some circular dependency could be tested in isolation. This is what was done for most of the basic underlying functions for the `proofSearch` algorithm. These tests can easily be automated by importing the test modules in the command line, and running the `:test` command. This executes all functions marked with the `test` keyword, which are boolean functions returning `true` if the test is successful and `false` otherwise. All unit tests in the code are provided in the `unitTests` submodule which can be found in appendix C.4.

In `ATP_Base_test.rsc` (listing C.4.1), unit tests have been written for both the `subTerm()` function, and the `fpLessThanOrEqualTo()` function. We test for whether a game is a subterm of itself, whether a game appears within a larger game, either directly, or nested more deeply, and test for both atomic and non-atomic games, and the order of subterms (for our purposes, order matters, so $a \wedge b$ is not a subterm of $b \wedge a$). Then we test whether `fpLessThanOrEqualTo()` only works on fixpoint formula and ignores the rest of the formula, taking only the game within the first modality into account.

In `ProofSearch_test.rsc` (listing C.4.2), unit tests were written for the `tryDisClo()` function (as it does not recursively call the `proofSearch()` function), the `detectCycles()` function, and the `proofSearchWeakExp()` function (since the input and output can easily be determined and tested). The `proofSearch()` algorithm itself is not unit tested, as it has circular dependencies with most of the `tryApply<Rule>()` functions. The unit tests for the `tryDisClo()` function test several combinations of current sequents and formulae saved in the `CloSeqs`. For `detectCycles()`, we test for when cycles should and should not be detected, we check for a variety of sequents, and lists of fixpoint sequents. And for the `proofSearchWeakExp()` function, we check several sequents, ones that are close together, and those who require many weak and/or exp rules, and possibly reordering of the terms and names in the labels, and ones that should not be possible to be reached through weakening and expanding.

In `RuleApplications_test.rsc` (listing C.4.3), there are unit tests present for each of the `apply<Rule>()` functions, plus the `tryApplyAx1()` function (as it does not recursively call the `proofSearch()` function). For most of the rules, we test with terms that are of the correct or incorrect format to apply the rule, and we even try applying the rules multiple times for some of the tests. For the `tryApplyAx1()` function we also try cases where weak or exp rules will have to be applied before the Ax1 can be applied. For the `applyIter()`, `applyDIter()`, and `applyClo()` functions, we also check for cases where the side condition should fail by providing certain `CloSeqs` values, and for the `applyClo()` function, we test whether the generated name is correct.

7.2 Manual Tests / Analysis

7.2.1 Setup

Of course, we also need to test whether our main `proofSearch()` algorithm works correctly. For this, an integration test has been written.

The `input` folder of the project contains a series of `.seq` files, which each contain a sequent that is inputted to the `proofSearch()` algorithm.

In the `integrationTests` submodule (appendix C.3), there is a single file `Proof_tests.rsc` (listing C.3.1) with a single `void` function `executeTests()`. This function calls the `proofSearch_Tool()` function for each of the sequents in the `input` folder. Thus, a `CloGProof` is generated in LaTeX for each of these input sequents, if a proof can be found. The proofs for these sequents can range from a single rule application, to proofs at least as big as the second example CloG proof in [22].

Appendix B.1 contains a table that goes over the resulting proofs for each of the sequents in the `input` folder, for each of the strategies (outlined in section 5.3) applied.

What follows is an analysis of these resulting proofs.

7.2.2 Sequents 1 – 5 (Simple Non-Valid Sequents)

The first five sequents are, respectively, the following (where formulae in the sequent are separated by commas, and full sequents are separated by semicolons): an empty sequent; p^ε; p^{x_0}; $p^\varepsilon, q^\varepsilon$; and p^{x_0,x_1}, q^{x_1,x_2}. None of these sequents are tautologies (fill in $p = \texttt{false}$ and $q = \texttt{false}$ and each of the sequents will be false), thus for none of these sequents, a proof should be found. And indeed, when using these sequents as an input to the program, no LaTeX proofs were generated for any of them.

7.2.3 Sequents 6 – 9 (Ax1)

The next four sequents test the `tryApplyAx1()` function. The first of these is just $p^\varepsilon, \neg p^\varepsilon$, which should be easily provable by just applying the axiom. This is indeed the proof that is found:

$$\dfrac{}{(p)^\varepsilon, (\neg p)^\varepsilon}\,\text{Ax1}$$

The other sequents are $p^{x_0,x_1}, \neg p^{x_1,x_2}$; $p^\varepsilon, \neg q^\varepsilon, r^\varepsilon, q^\varepsilon$; and $p^\varepsilon, \neg q^{x_1,x_2}, r^{x_1}, q^{x_0}$ respectively. These sequents can easily be proven by weakening and expanding before applying the axiom. Here we show the proof for $p^\varepsilon, \neg q^{x_1,x_2}, r^{x_1}, q^{x_0}$. The remaining proofs can be found in appendix B.1.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg q)^\varepsilon, (q)^\varepsilon}\,\text{Ax1}}{(\neg q)^\varepsilon, (q)^{x_0}}\,\text{exp}}{(\neg q)^\varepsilon, (r)^{x_1}, (q)^{x_0}}\,\text{weak}}{(\neg q)^{x_2}, (r)^{x_1}, (q)^{x_0}}\,\text{exp}}{(\neg q)^{x_1,x_2}, (r)^{x_1}, (q)^{x_0}}\,\text{exp}}{(p)^\varepsilon, (\neg q)^{x_1,x_2}, (r)^{x_1}, (q)^{x_0}}\,\text{weak}$$

7.2.4 Sequent 10 (Closure Sequent Discharge)

The tenth test consists of multiple sequents, as we are testing the closure sequent discharge. One sequent, $\langle a\rangle p^{x_0}$ is used as the current sequent, and another, $\langle a\rangle p^\varepsilon$, is used as a saved closure sequent with the name x_0, and as a saved fixpoint sequent for cycle detection. A closure sequent discharge should be possible immediately. And it is, as seen in the resulting single-line proof:

$$[(\langle a^\times\rangle p)^{x_0}]^{x_0}$$

7.2.5 Sequents 11 & 12 (Or Rule)

Next up are test cases for the $\vee$ rule: $(p \vee \neg p)^\varepsilon$; and $p^{x_0}, (q \vee \neg q)^{x_0,x_1}, r^{x_2}$. Both of these end in a `Ax1` application, but the latter requires some weakening and expanding first. We can see these proofs in figure 2.

$$\dfrac{\dfrac{}{(p)^\varepsilon, (\neg p)^\varepsilon}\,\text{Ax1}}{(p \vee \neg p)^\varepsilon}\,\vee$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(q)^\varepsilon, (\neg q)^\varepsilon}\,\text{Ax1}}{(q)^\varepsilon, (\neg q)^\varepsilon, (r)^{x_2}}\,\text{weak}}{(q)^\varepsilon, (\neg q)^{x_1}, (r)^{x_2}}\,\text{exp}}{(q)^\varepsilon, (\neg q)^{x_0,x_1}, (r)^{x_2}}\,\text{exp}}{(q)^{x_1}, (\neg q)^{x_0,x_1}, (r)^{x_2}}\,\text{exp}}{(q)^{x_0,x_1}, (\neg q)^{x_0,x_1}, (r)^{x_2}}\,\text{exp}}{(p)^{x_0}, (q)^{x_0,x_1}, (\neg q)^{x_0,x_1}, (r)^{x_2}}\,\text{weak}}{(p)^{x_0}, (q \vee \neg q)^{x_0,x_1}, (r)^{x_2}}\,\vee$$

Figure 2: proofs for the sequents $(p \vee \neg p)^\varepsilon$ on the left and $p^{x_0}, (q \vee \neg q)^{x_0,x_1}, r^{x_2}$ on the right

7.2.6 Sequents 13 – 19 (Other Local Rules)

Sequents 13 up to 19 are simple testcases for specific rules, the $\wedge$, mod_m, $\sqcup$, $\sqcap$, ;, ?, and ! rules respectively. To form a valid proof and end in a $\mathsf{Ax1}$ axiom application, some of the resulting proofs need to apply some different rules (for example, the $\vee$ rule is applied after the $\sqcup$ rule, and the mod_m rule is applied after game connectives $\sqcup$, $\sqcap$, and ;). The proofs resulting from these sequents can be found in figure 3.

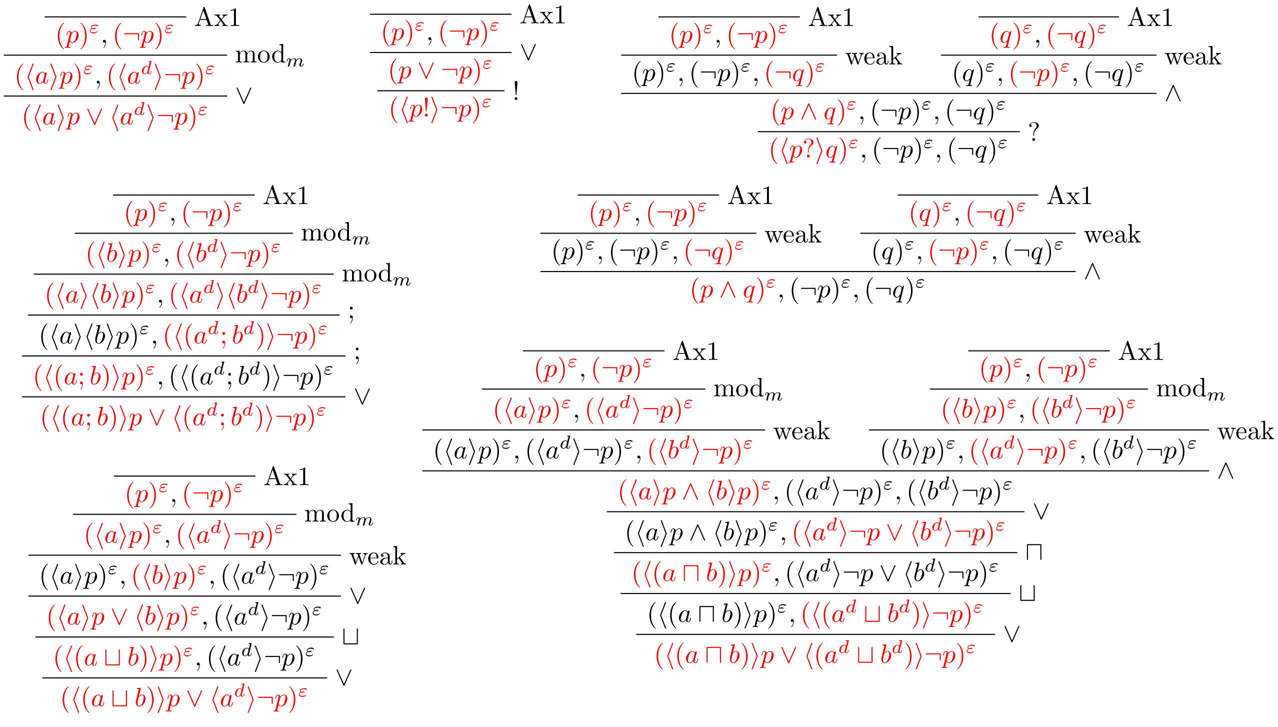


Figure 3: various proofs found for sequents testing specific rules

7.2.7 Sequent 20 (Monotonicity)

Next, we test a general property of game logic. Since we know monotonicity must hold for Game Logic, the formula $\langle a\rangle p \to \langle a\rangle(p \vee q)$ should be valid. We can convert this to $\mathcal{L}_{\mathsf{NF}}$, such that we obtain the formula $\langle a^d\rangle\neg p \vee \langle a\rangle(p \vee q)$. This corresponds to the CloG sequent $(\langle a^d\rangle\neg p \vee \langle a\rangle(p \vee q))^\varepsilon$, for which we obtain the following proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{(\neg p)^\varepsilon, (p)^\varepsilon}\ \text{Ax1}}{(\neg p)^\varepsilon, (p)^\varepsilon, (q)^\varepsilon}\ \text{weak}}{(\neg p)^\varepsilon, (p \vee q)^\varepsilon}\ \vee}{(\langle a^d\rangle\neg p)^\varepsilon, (\langle a\rangle(p \vee q))^\varepsilon}\ \text{mod}_m}{(\langle a^d\rangle\neg p \vee \langle a\rangle(p \vee q))^\varepsilon}\ \vee$$

7.2.8 Sequents 21 – 24 (Least Fixpoint Formulae)

Next up, we want to test the $*$ rule. We know that if a proposition p holds, then Angel has a strategy to ensure p holds after playing game a a certain number of times, namely 0 times. Thus the formula $p \to \langle a^*\rangle p$ must hold. Converting this to $\mathcal{L}_{\mathsf{NF}}$ and adding an empty label, we get the sequent $(\neg p \vee \langle a^*\rangle p)^\varepsilon$.

Similarly, we can reason that if Angel has a strategy to ensure p holds after playing game a once, then Angel has a strategy to ensure p holds after playing game a a certain number of times. $\langle a\rangle p \to \langle a^*\rangle p$ must also hold. We try finding a proof with the sequent $(\langle a^d\rangle\neg p \vee \langle a^*\rangle p)^\varepsilon$.

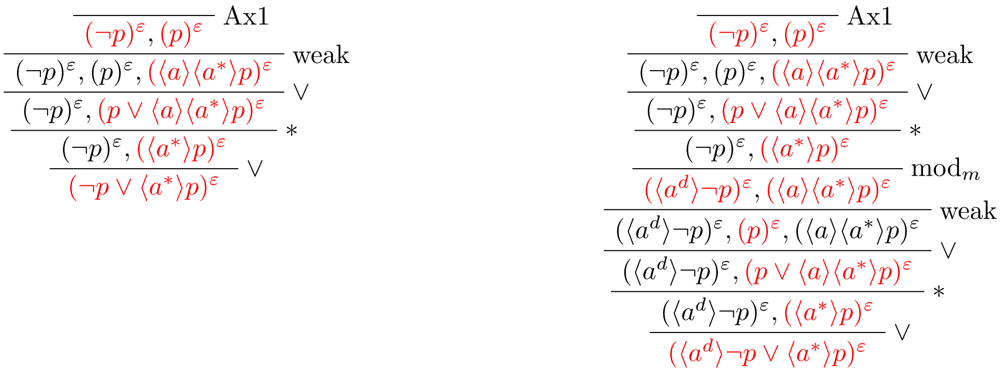

Ax1
$(\neg p)^\varepsilon, (p)^\varepsilon$
weak
$(\neg p)^\varepsilon, (p)^\varepsilon, (\langle a\rangle\langle a^*\rangle p)^\varepsilon$
$\vee$
$(\neg p)^\varepsilon, (p \vee \langle a\rangle\langle a^*\rangle p)^\varepsilon$
$*$
$(\neg p)^\varepsilon, (\langle a^*\rangle p)^\varepsilon$
$\vee$
$(\neg p \vee \langle a^*\rangle p)^\varepsilon$
Ax1
$(\neg p)^\varepsilon, (p)^\varepsilon$
weak
$(\neg p)^\varepsilon, (p)^\varepsilon, (\langle a\rangle\langle a^*\rangle p)^\varepsilon$
$\vee$
$(\neg p)^\varepsilon, (p \vee \langle a\rangle\langle a^*\rangle p)^\varepsilon$
$*$
$(\neg p)^\varepsilon, (\langle a^*\rangle p)^\varepsilon$
mod_m
$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a\rangle\langle a^*\rangle p)^\varepsilon$
weak
$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a\rangle\langle a^*\rangle p)^\varepsilon$
$\vee$
$(\langle a^d\rangle\neg p)^\varepsilon, (p \vee \langle a\rangle\langle a^*\rangle p)^\varepsilon$
$*$
$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a^*\rangle p)^\varepsilon$
$\vee$
$(\langle a^d\rangle\neg p \vee \langle a^*\rangle p)^\varepsilon$

Figure 4: proofs for the sequents $(\neg p \vee \langle a^*\rangle p)^\varepsilon$ on the left and $(\langle a^d\rangle\neg p \vee \langle a^*\rangle p)^\varepsilon$ on the right

The proofs for these two sequents can be found in figure 4.

As can be seen, these proofs are similar. The sequence of rules applied to the first proof, is applied twice to the second proof. The fixpoint is unfolded, an $\vee$ rule is applied, followed by a weak rule. At that point for the second proof, we can apply a mod_m, such that we essentially end up in the situation where Angel has played the game once, and we must once again prove the previous sequent, $(\neg p \vee \langle a^*\rangle p)^\varepsilon$. And once again, we apply the $*$, $\vee$, and weak rule before being able to apply the Ax1 axiom.

Another test for the $*$ rule uses the fact that $\langle a^{**}\rangle$ has the same meaning as $\langle a^*\rangle$. $\langle a^{**}\rangle$ just means Angel has a strategy of ensuring p after playing a game, which Angel chooses to play a certain number of times, a certain number of times. That is not any different to just playing a game a certain number of times.

Thus, since $\langle a\rangle p \rightarrow \langle a^*\rangle p$ holds, so must $\langle a\rangle p \rightarrow \langle a^{**}\rangle p$. We try finding a proof with the sequent $(\langle a^d\rangle\neg p \vee \langle a^{**}\rangle p)^\varepsilon$, which should be interesting since it involves nested fixpoints.

As described in section 5.3, there could be multiple ways of ordering the rule applications. Up to this point, the resulting proofs have been the same regardless of which strategy was applied. However, for this next proof, there is a slight difference in the resulting proofs.

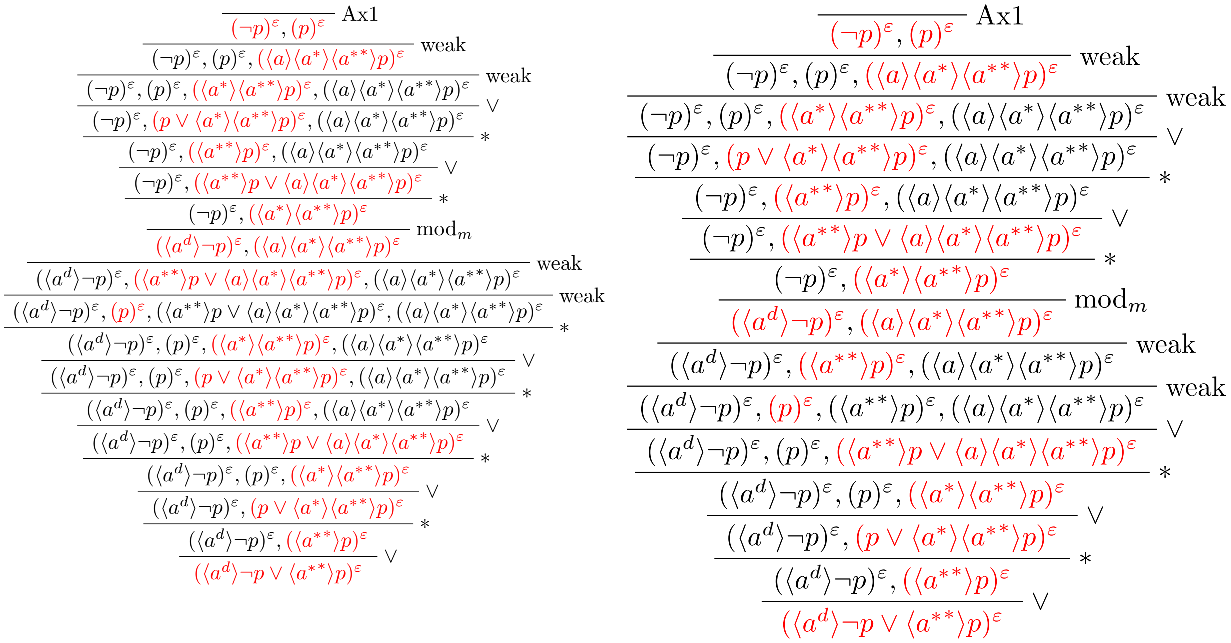

Ax1
$(\neg p)^\varepsilon, (p)^\varepsilon$
weak
$(\neg p)^\varepsilon, (p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
weak
$(\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$\vee$
$(\neg p)^\varepsilon, (p \vee \langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$*$
$(\neg p)^\varepsilon, (\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$\vee$
$(\neg p)^\varepsilon, (\langle a^{**}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$*$
$(\neg p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
mod_m
$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
weak
$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a^{**}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
weak
$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{**}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$*$
$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$\vee$
$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (p \vee \langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$*$
$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$\vee$
$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{**}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$*$
$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$\vee$
$(\langle a^d\rangle\neg p)^\varepsilon, (p \vee \langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$*$
$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a^{**}\rangle p)^\varepsilon$
$\vee$
$(\langle a^d\rangle\neg p \vee \langle a^{**}\rangle p)^\varepsilon$
Ax1
$(\neg p)^\varepsilon, (p)^\varepsilon$
weak
$(\neg p)^\varepsilon, (p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
weak
$(\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$\vee$
$(\neg p)^\varepsilon, (p \vee \langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$*$
$(\neg p)^\varepsilon, (\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$\vee$
$(\neg p)^\varepsilon, (\langle a^{**}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$*$
$(\neg p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
mod_m
$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
weak
$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
weak
$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{**}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$\vee$
$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{**}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$*$
$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$\vee$
$(\langle a^d\rangle\neg p)^\varepsilon, (p \vee \langle a^*\rangle\langle a^{**}\rangle p)^\varepsilon$
$*$
$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a^{**}\rangle p)^\varepsilon$
$\vee$
$(\langle a^d\rangle\neg p \vee \langle a^{**}\rangle p)^\varepsilon$

Figure 5: proofs for the sequent $(\langle a^d\rangle\neg p \vee \langle a^{**}\rangle p)^\varepsilon$, on the left for the Procrastination or Simple clo Strategy, and on the right for the Pre-Emptive mod_m or Greedy Strategy

As can be seen from figure 5, using a strategy that applies the mod_m rule as late as possible, results in a longer proof. The mod_m rule could have already been applied after the second unfolding and $\vee$ rule application. However, it is not until two unfoldings later that a cycle is detected, and the sequent is fully saturated, thus the mod_m is tried. If instead, we try to apply the mod_m as soon as possible, we find this shorter proof on the right-hand side of the image.

For sequent 24, this difference is even more clear. We add another fixpoint operator, and try to find a proof for $\langle a\rangle p \to \langle a^{***}\rangle p$, corresponding to the sequent $(\langle a^d\rangle\neg p \vee \langle a^{***}\rangle p)^\varepsilon$. We show the resulting proof for the Pre-Emptive mod_m Strategy or Greedy Strategy (the other proof can be found in appendix B.1):

$$\begin{prooftree}
\AxiomC{}
\RightLabel{Ax1}
\UnaryInfC{$(\neg p)^\varepsilon, (p)^\varepsilon$}
\RightLabel{weak}
\UnaryInfC{$(\neg p)^\varepsilon, (p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{weak}
\UnaryInfC{$(\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{weak}
\UnaryInfC{$(\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$\vee$}
\UnaryInfC{$(\neg p)^\varepsilon, (p \vee \langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$*$}
\UnaryInfC{$(\neg p)^\varepsilon, (\langle a^{***}\rangle p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$\vee$}
\UnaryInfC{$(\neg p)^\varepsilon, (\langle a^{***}\rangle p \vee \langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$*$}
\UnaryInfC{$(\neg p)^\varepsilon, (\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$\vee$}
\UnaryInfC{$(\neg p)^\varepsilon, (\langle a^{**}\rangle\langle a^{***}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$*$}
\UnaryInfC{$(\neg p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{mod_m}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{weak}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{weak}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{weak}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a^{***}\rangle p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{weak}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{***}\rangle p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$\vee$}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{***}\rangle p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a^{**}\rangle\langle a^{***}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$\vee$}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{***}\rangle p \vee \langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a^{**}\rangle\langle a^{***}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$*$}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{***}\rangle p \vee \langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$*$}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$\vee$}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (p \vee \langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$*$}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{***}\rangle p)^\varepsilon, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$\vee$}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{***}\rangle p \vee \langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$*$}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$\vee$}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (p \vee \langle a^{**}\rangle\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$*$}
\UnaryInfC{$(\langle a^d\rangle\neg p)^\varepsilon, (\langle a^{***}\rangle p)^\varepsilon$}
\RightLabel{$\vee$}
\UnaryInfC{$(\langle a^d\rangle\neg p \vee \langle a^{***}\rangle p)^\varepsilon$}
\end{prooftree}$$

Admittedly, even this proof is not as efficient as it could be. The third $*$ rule application from the bottom was unnecessary. The same fixpoint formula $(\langle a^{***}\rangle p^\varepsilon)$ was already unfolded at the first $*$ rule application from the bottom. There might be additional heuristics to prevent this sort of thing, such as not unfolding the same fixpoint formula multiple times before a mod_m application.

7.2.9 Sequents 25 – 27 (Greatest Fixpoint Formulae)

Now, obviously the proposition $\langle a^*\rangle p \to \langle a^*\rangle p$ holds, but if we rewrite this proposition in $\mathcal{L}_{\mathsf{NF}}$, we get $\langle a^{d^\times}\rangle\neg p \vee \langle a^*\rangle p$. Sequent 25 is the sequent containing this formula and we want to test it since it involves a greatest fixpoint formula.

Using the Procrastination or Pre-Emptive mod_m Strategy, where we apply the clo rule as late as possible, we obtain the proof in figure 6.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\,\text{Ax1}
}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\text{weak}
}{(\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\text{exp}
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{[(\langle a^{d^\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}]^{x_0}}{(\langle a^{d^\times}\rangle\neg p)^{x_0}, (p \vee \langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\vee
}{(\langle a^{d^\times}\rangle\neg p)^{x_0}, (\langle a^*\rangle p)^{\varepsilon}}\,*
}{(\langle a^d\rangle\langle a^{d^\times}\rangle\neg p)^{x_0}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\text{mod}_m
}{(\langle a^d\rangle\langle a^{d^\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\text{weak}
}{(\neg p \wedge \langle a^d\rangle\langle a^{d^\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\wedge
}{(\langle a^{d^\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\text{clo}_{x_0}
}{(\langle a^{d^\times}\rangle\neg p)^{\varepsilon}, (p \vee \langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\vee
}{(\langle a^{d^\times}\rangle\neg p)^{\varepsilon}, (\langle a^*\rangle p)^{\varepsilon}}\,*
}{(\langle a^{d^\times}\rangle\neg p \vee \langle a^*\rangle p)^{\varepsilon}}\,\vee
$$

Figure 6: proof for $\langle a^{d^\times}\rangle\neg p \vee \langle a^*\rangle p$, using the Procrastination or Pre-Emptive mod_m Strategy; contains 12 rule applications

The last two rule applications on the right branch in this proof are the $*$ and $\vee$ rule, which also occur before the clo rule application. We can imagine swapping the clo rule application with the $\vee$ rule application below it, which makes the sequent to match in the clo rule assumption the same as what is currently the sequent below the upper $\vee$ rule application. This means we need one less $\vee$ rule application to match this assumption. We can make the same argument again, but this time swapping the clo rule application with the $*$ rule application.

In this specific instance, we can move the clo rule application *down* twice. Perhaps it is not surprising then that we obtain a different proof if we try use the Simple clo or Greedy Strategy, where we apply the clo rule as early as possible. Figure 7 shows the proof for these strategies.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\,\text{Ax1}
}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\text{weak}
}{(\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\text{exp}
}{(\neg p)^{x_0}, (p \vee \langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\vee
}{(\neg p)^{x_0}, (\langle a^*\rangle p)^{\varepsilon}}\,*
\qquad
\dfrac{
\dfrac{
\dfrac{
\dfrac{[(\langle a^{d^\times}\rangle\neg p)^{x_0}, (\langle a^*\rangle p)^{\varepsilon}]^{x_0}}{(\langle a^d\rangle\langle a^{d^\times}\rangle\neg p)^{x_0}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\text{mod}_m
}{(\langle a^d\rangle\langle a^{d^\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\text{weak}
}{(\langle a^d\rangle\langle a^{d^\times}\rangle\neg p)^{x_0}, (p \vee \langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\,\vee
}{(\langle a^d\rangle\langle a^{d^\times}\rangle\neg p)^{x_0}, (\langle a^*\rangle p)^{\varepsilon}}\,*
}{(\neg p \wedge \langle a^d\rangle\langle a^{d^\times}\rangle\neg p)^{x_0}, (\langle a^*\rangle p)^{\varepsilon}}\,\wedge
}{(\langle a^{d^\times}\rangle\neg p)^{\varepsilon}, (\langle a^*\rangle p)^{\varepsilon}}\,\text{clo}_{x_0}
}{(\langle a^{d^\times}\rangle\neg p \vee \langle a^*\rangle p)^{\varepsilon}}\,\vee
$$

Figure 7: proof for $\langle a^{d^\times}\rangle\neg p \vee \langle a^*\rangle p$, using the Simple clo or Greedy Strategy; contains 12 rule applications

The proof has the same number of rule applications as the one found with the other strategies. The last $*$ and $\vee$ application on the right branch were not necessary anymore, since the clo rule assumption is matched earlier. But another $*$ and $\vee$ rule application have been added to the left branch, since the $\wedge$ rule is applied before the $*$ rule.

The next sequent we try contains both nested fixpoint formulae, as well as a greatest fixpoint formula. It is derived from the proposition $\langle a^{\times^\times}\rangle p \to \langle a^\times\rangle p$. This is true because $a^{\times^\times}$ is equivalent to $a^\times$ (if Angel has a strategy to ensure some state holds after playing a game, that is played any number of times, any number of times, that is the same as just saying they have a strategy to ensure some state holds after playing that game any number of times). We obtain the sequent in normal form: $(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{\times}\rangle p)^{\varepsilon}$.

Using the Procrastination Strategy, we obtain the proof in figure 8. If we instead use the Pre-Emptive mod_m Strategy (see figure 9), we can reduce the size of the proof. Instead of 26 rule applications, the resulting proof only contains 23. We will skip the simple clo Strategy, but it can be found in appendix B.1. Interestingly enough, trying the Greedy Strategy results in a larger proof again (30 rule applications, see figure 10).

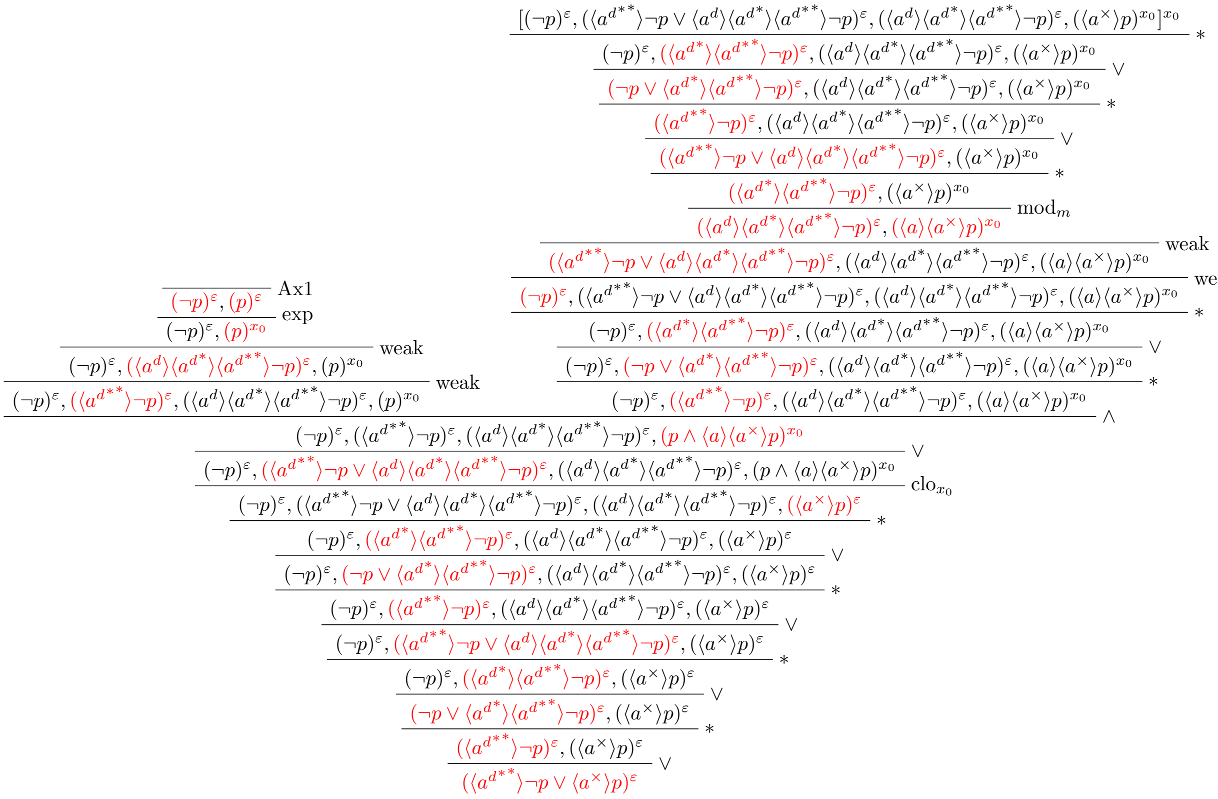

$[(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}]^{x_0}$
$*$
$(\neg p)^{\varepsilon}, (\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}$
$\vee$
$(\neg p \vee \langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}$
$*$
$(\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}$
$\vee$
$(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}$
$*$
$(\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}$
mod_m
$(\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}$
weak
$(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}$
weak
$(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}$
$*$
$(\neg p)^{\varepsilon}, (\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}$
$\vee$
$(\neg p)^{\varepsilon}, (\neg p \vee \langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}$
$*$
$(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}$
$(\neg p)^{\varepsilon}, (p)^{\varepsilon}$
Ax1
$(\neg p)^{\varepsilon}, (p)^{x_0}$
exp
$(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (p)^{x_0}$
weak
$(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (p)^{x_0}$
weak
$\wedge$
$(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (p \wedge \langle a\rangle\langle a^{\times}\rangle p)^{x_0}$
$\vee$
$(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (p \wedge \langle a\rangle\langle a^{\times}\rangle p)^{x_0}$
clo_{x_0}
$(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}$
$*$
$(\neg p)^{\varepsilon}, (\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}$
$\vee$
$(\neg p)^{\varepsilon}, (\neg p \vee \langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}$
$*$
$(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}$
$\vee$
$(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}$
$*$
$(\neg p)^{\varepsilon}, (\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}$
$\vee$
$(\neg p \vee \langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}$
$*$
$(\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}$
$\vee$
$(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{\times}\rangle p)^{\varepsilon}$

Figure 8: proof for $(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{\times}\rangle p)^{\varepsilon}$, using the Procrastination Strategy; contains 26 rule applications

The size of the proof in figure 10 compared to the one in figure 8 could be explained as follows: in the proof in figure 10, the assumption for the clo rule could not be matched initially, and the greatest fixpoint needed to be unfolded *again*. And only for this second clo rule application, the assumption was matched, thus in the post-processing phase (see section 6.2.7), the first clo rule application was replaced by a × application instead.

An even smaller proof than the one in figure 10 could have been found, if after the mod_m application, the $*$ rule would have been tried before the clo rule. We would have obtained the sequent $(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, \langle a^{\times}\rangle p^{\varepsilon}$, to which we could apply disjunction, resulting in $\langle a^{d^{**}}\rangle\neg p^{\varepsilon}, \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p^{\varepsilon}, \langle a^{\times}\rangle p^{\varepsilon}$, and then weakening on the second term, resulting in $\langle a^{d^{**}}\rangle\neg p^{\varepsilon}, \langle a^{\times}\rangle p^{\varepsilon}$, which then matches the conclusion of the initial clo rule application (which in this proof has been replaced by the × rule application), thus can be discharged.

This proof would have only unfolded the greatest fixpoint once, just like the proofs found with the Procrastination and Pre-Emptive mod_m Strategies. Perhaps it might be possible to generalize this transformation, maybe by altering the order of rules after a mod_m rule application, or by a more goal-directed desired cycle detection (where we actively try to reach the sequent that was the assumption for an earlier applied clo rule).

Adding another level of complexity (with a double angelic as well as demonic iteration), since $\langle a^{**}\rangle p \to \langle a^{**}\rangle p$ is true, we can also try the sequent with the formula $\langle a^{d^{\times\times}}\rangle\neg p \vee \langle a^{**}\rangle p$.

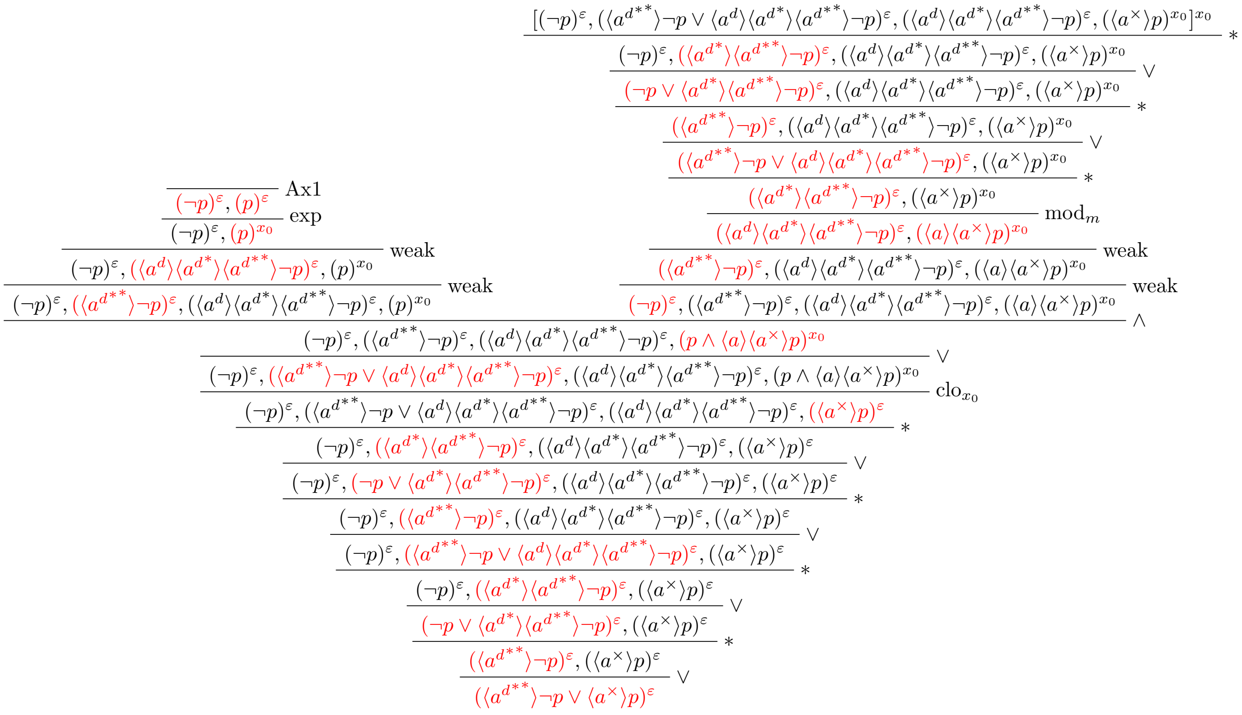


Figure 9: proof for $(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{\times}\rangle p)^{\varepsilon}$, using the Pre-Emptive mod_m Strategy; contains 23 rule applications

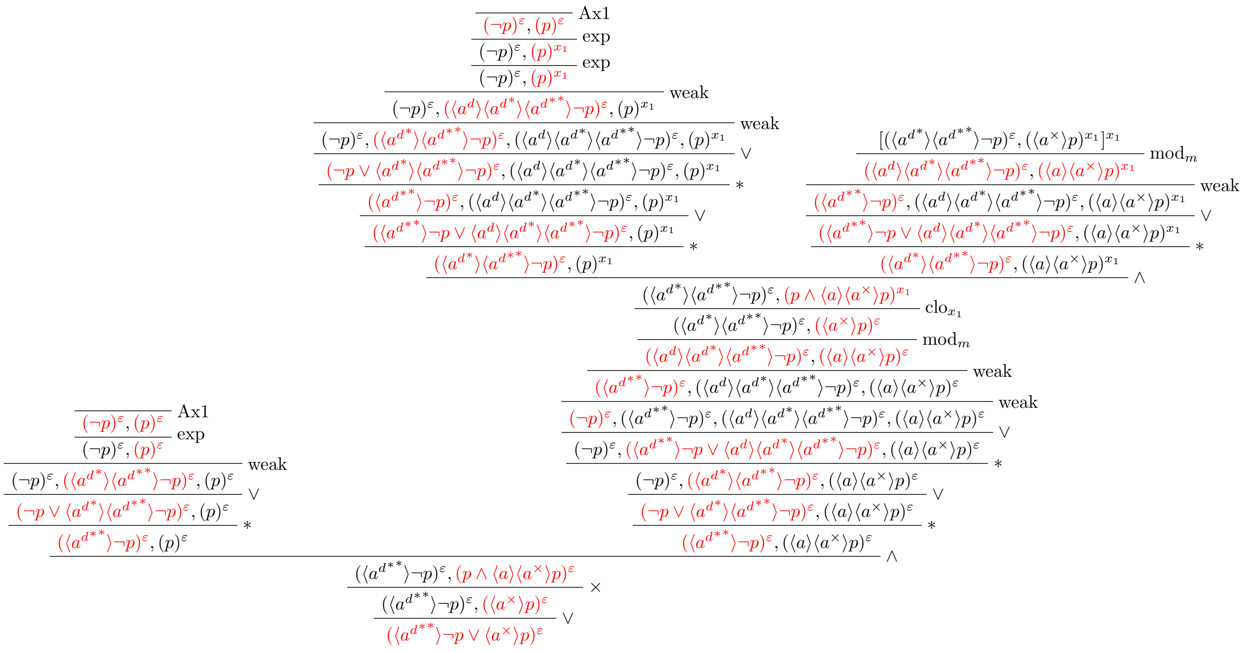


Figure 10: proof for $(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{\times}\rangle p)^{\varepsilon}$, using the Greedy Strategy; contains 30 rule applications

Skipping the Procrastination Strategy for now (which once again, only contains 3 more rule applications than the Pre-Emptive mod_m Strategy), we show the proof found by the Pre-Emptive mod_m Strategy in figure 11.

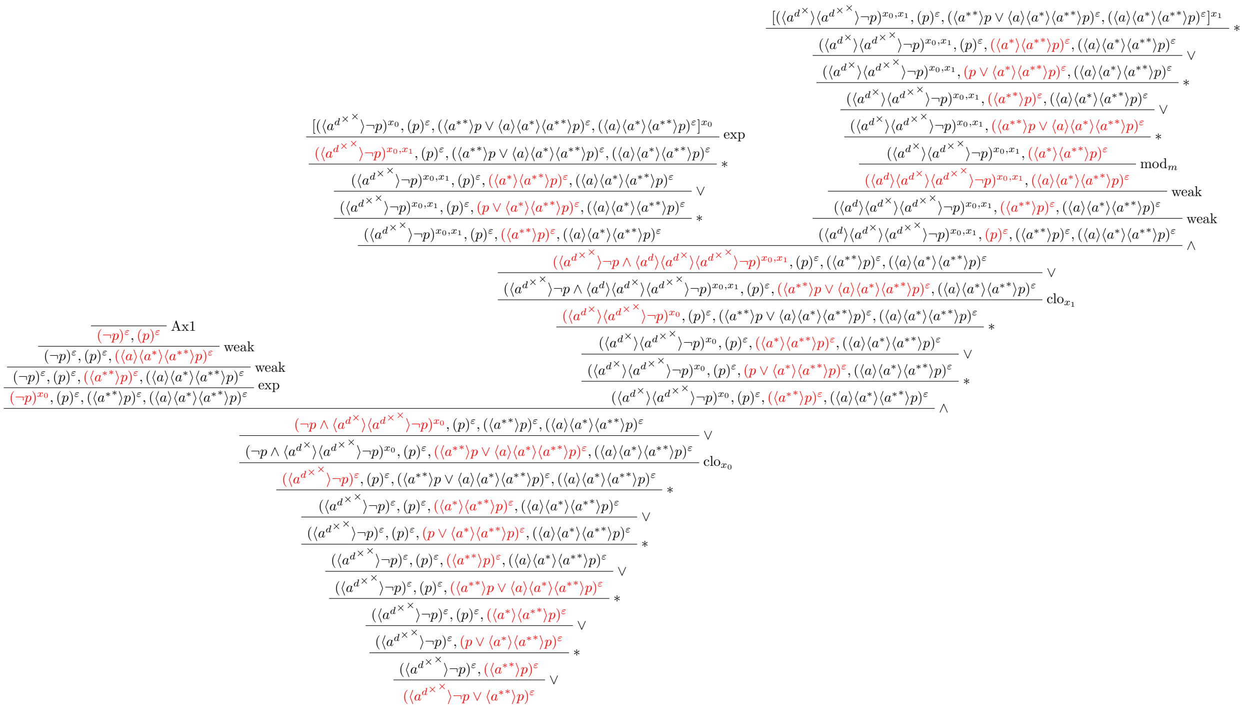


Figure 11: proof for $\langle a^{d^{\times\times}}\rangle\neg p \vee \langle a^{**}\rangle p$, using the Pre-Emptive mod_m Strategy; contains 33 rule applications

We should note that this proof already contains two clo rule applications. Once again, however, applying the Greedy Strategy (see figure 12) adds another clo rule application (although it has once again been replaced by a × rule application):

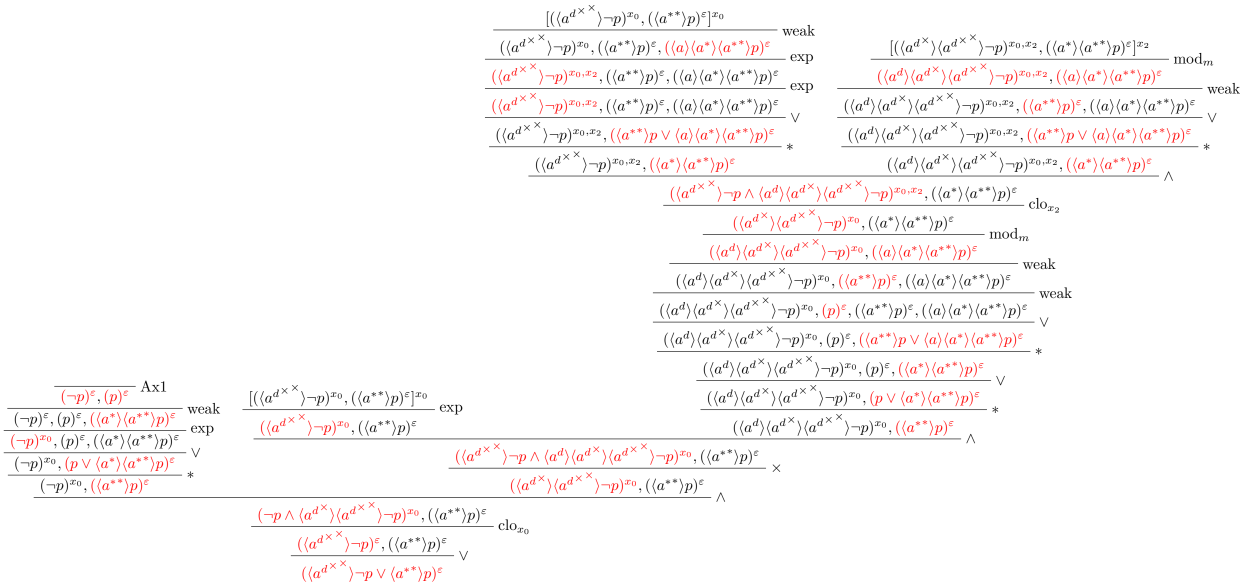


Figure 12: proof for $\langle a^{d^{\times\times}}\rangle\neg p \vee \langle a^{**}\rangle p$, using the Greedy Strategy; contains 29 rule applications

Contrary to the last example, this proof is in fact smaller than the one obtained with the Simple clo Rule

Strategy. It only contains 29, as opposed to 32 rule applications. More testing would have to be done, but it appears the Simple clo and Greedy strategies find shorter proofs for more complicated sequents with more nested fixpoint formulae.

Once again, if after the mod_m rule application, we would have applied the $*$ rule, then the $\vee$ rule, and the weak rule, we would have been able to match the conclusion of this $\times$ (former clo) rule, and the proof would have been shorter. See section 8.2 for a further discussion of this strategy.

7.2.10 Sequents 28 – 30 (Many Nested Fixpoint Formulae)

The next sequents to try contain many nested fixpoints. For the first of these, since we know $p \to \langle a^* \rangle p$, it must also be the case that $p \to \langle a^{******} \rangle p$, thus the sequent $(\neg p \vee \langle a^{******} \rangle) p^\varepsilon$ must be valid. What follows is the proof for this sequent:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^\varepsilon, (p)^\varepsilon}\,\text{Ax1}}{(\neg p)^\varepsilon, (p)^\varepsilon, (\langle a^{*****} \rangle \langle a^{******} \rangle p)^\varepsilon}\,\text{weak}}{(\neg p)^\varepsilon, (p \vee \langle a^{*****} \rangle \langle a^{******} \rangle p)^\varepsilon}\,\vee}{(\neg p)^\varepsilon, ((\langle a^{******} \rangle p)^\varepsilon}\,*}{(\neg p \vee \langle a^{******} \rangle p)^\varepsilon}\,\vee$$

This proof is quite small, since only one fixpoint needed to be unfolded to find it. The same proof is found regardless of the mod_m and clo rule ordering. And since the $\vee$ rule application and axiom matching is tried before applying another $*$ rule, the formula was not unfolded further.

A more interesting proof can be found for the proposition $\langle a \rangle p \to \langle a^{******} \rangle p$, or corresponding sequent $(\langle a^d \rangle \neg p \vee \langle a^{******} \rangle p)^\varepsilon$. The proof found is too big to properly display using LaTeX. A scaled-down image of the proof can be seen in figure 13.

Trying to apply the mod_m rule as early as possible already reduces the size of the proof dramatically, as can also be seen in figure 13.

These proofs are massive, since the fixpoint formula needs to be unfolded completely in order to reach the axiom. That being said, there are more efficient proofs for this sequent. The proofs found still do many unnecessary unfoldings.

For this example, we can think of a strategy that can greatly reduce the size of the resulting proof and limit the search space. This strategy involves weakening the sequent after each $\vee$ rule application. We essentially "force" the algorithm to pick one of the resulting formulae of the $\vee$ rule application, which makes it so the resulting proof does not propagate unnecessary side formulae, which then also cannot be unfolded any further. We should not weaken after the first $\vee$ rule application, however, since we need both of the resulting formulae. When we do allow this, the program produces a stack overflow. But if we instead start with the sequent $\langle a^d \rangle \neg p^\varepsilon, \langle a^{******} \rangle p^\varepsilon$, and we implement this weakening after $\vee$ application, we get the proof in figure 14.

Similar to the last example, we know that $\langle a^\times \rangle p \to \langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}} \rangle p$. Thus, we can try the sequent $(\langle a^{d^*} \rangle \neg p \vee \langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}} \rangle p)^\varepsilon$.

The resulting proof for the Simple clo or Greedy Strategy can be found in figure 15.

In the resulting proof, each of the 6 greatest fixpoint formulae is unfolded which creates many branches. The proof for the other strategies can be found in appendix B.1. Also, we show both proofs in more detail in appendix B.2, from which we can see that for example each of the branches is closed by a separate clo rule assumption.

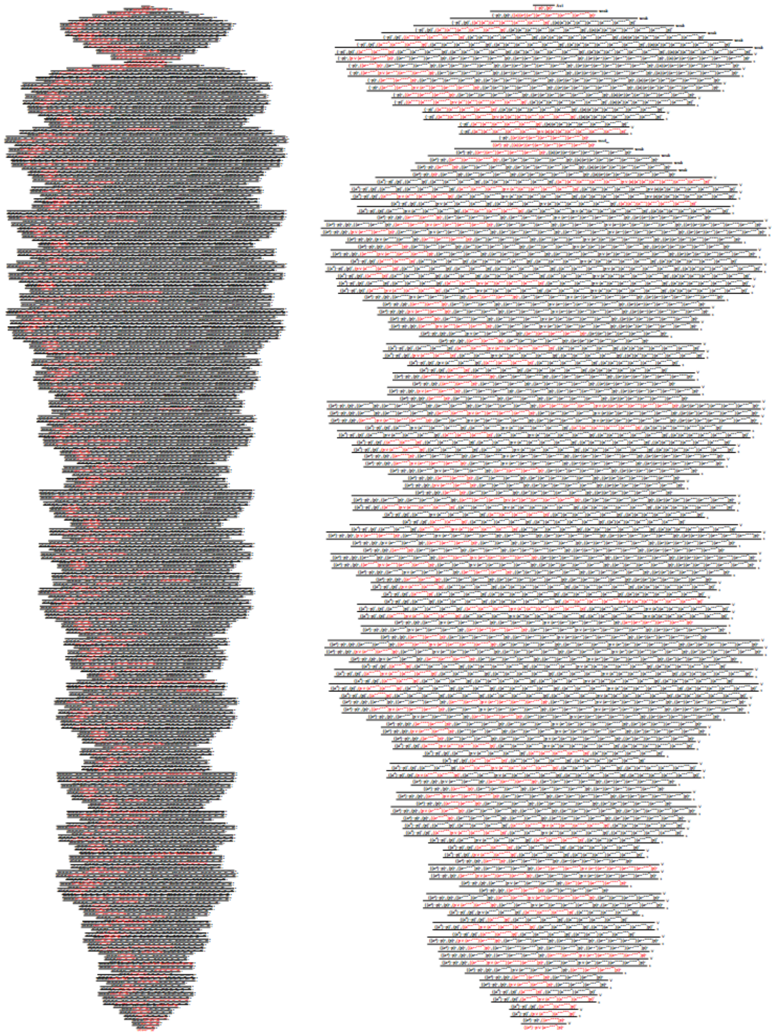


Figure 13: monster proofs for the sequent $\langle a^{d}\rangle\neg p \vee \langle a^{*^{*^{*^{*^{*^{*}}}}}}\rangle p^{\varepsilon}$; on the left, the Procrastination or Simple clo Strategy was applied, whereas on the right, the Pre-Emptive mod_m or Greedy Strategy was applied

7.2.11 Sequents 31 – 35 (Non-Valid Nested Fixpoints)

Proofs for sequents 31 to 35 do not exist. These sequents instead test whether multiple nested fixpoint operators cause an undesired cycle to not be detected such that the algorithm gets stuck in infinite recursion, or whether it will take an unreasonable amount of time to complete the proof search. The sequents that are tested are $\langle a^{*^{*}}\rangle p^{\varepsilon}$; $\langle a^{\times^{\times}}\rangle p^{\varepsilon}$; $\langle a^{*^{*^{*^{*^{*^{*}}}}}}\rangle p^{\varepsilon}$; $\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p^{\varepsilon}$; and $\langle a^{*^{\times^{*^{\times^{*^{\times}}}}}}\rangle p^{\varepsilon}$, respectively. Most of these return within a fraction of a second that there is no proof. Only $\langle a^{*^{*^{*^{*^{*^{*}}}}}}\rangle p^{\varepsilon}$ takes a few seconds instead. There does not seem to be a significant difference in execution time depending on which strategy is used.

Something interesting could be said for the sequent $\langle a^{*^{\times^{*^{\times^{*^{\times}}}}}}\rangle p^{\varepsilon}$. An earlier implementation of the program allowed for the application of × rules, and could lead to particular undesired cycles that were not detected by the cycle detection for this sequent. Figure 16 illustrates these kind of undesired cycles for a slightly smaller sequent $\langle a^{\times^{*^{\times}}}\rangle p^{\varepsilon}$. As can be seen, the sequents marked in blue are the same sequent with an extra formula added each time, but these were not detected by the cycle detection. We could fix this problem by checking for supersets of earlier occurred sequents in our cycle detection, but this would cause problems for proofs of different sequents. Instead, we conjecture that trying to apply the × rule is unnecessary to find a proof, and this problem was solved by not allowing × rule applicatons. More on this in section 8.2.

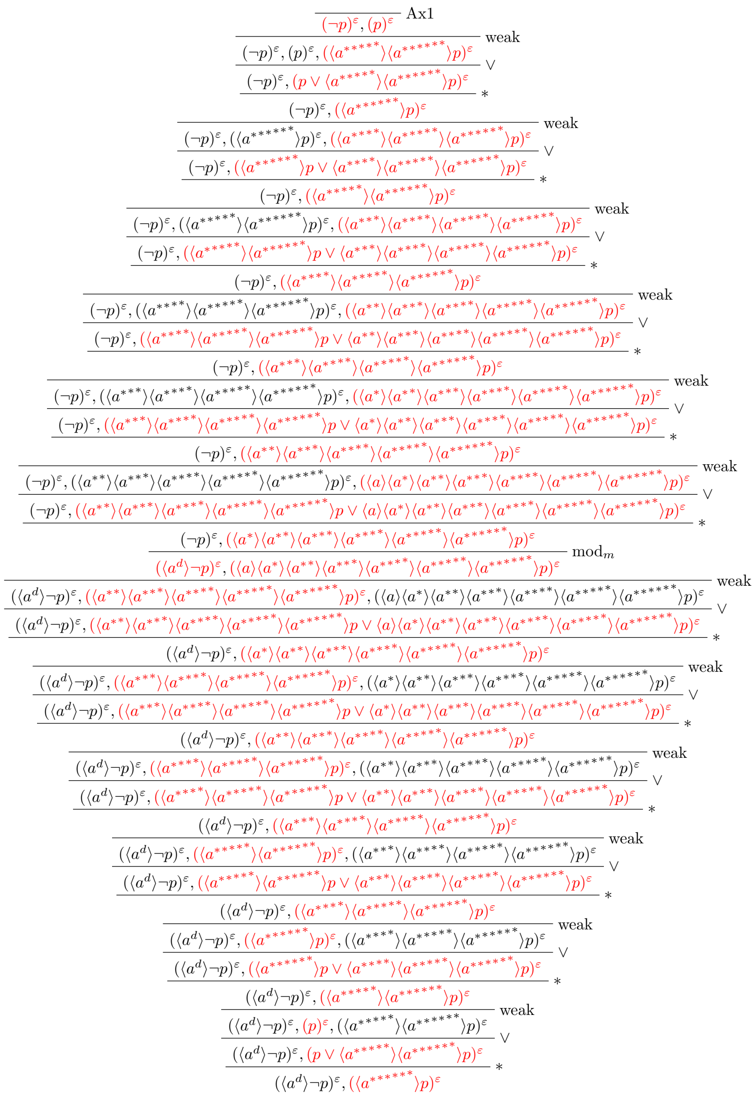


Figure 14: proof for $\langle a^d\rangle\neg p^\varepsilon, \langle a^{******}\rangle p^\varepsilon$ using a custom “weakening after $\vee$ application” strategy

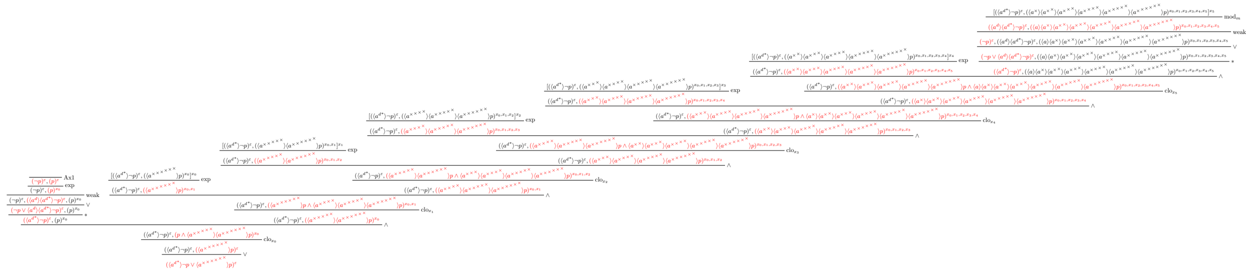


Figure 15: proof for $(\langle a^{d^*}\rangle\neg p \vee \langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^\varepsilon$, using the Simple clo or Greedy Strategy; contains 27 rule applications

7.2.12 Sequents 36 – 39 (Complex Sequents)

The last 4 testcases are for some larger, non-trivial sequents.

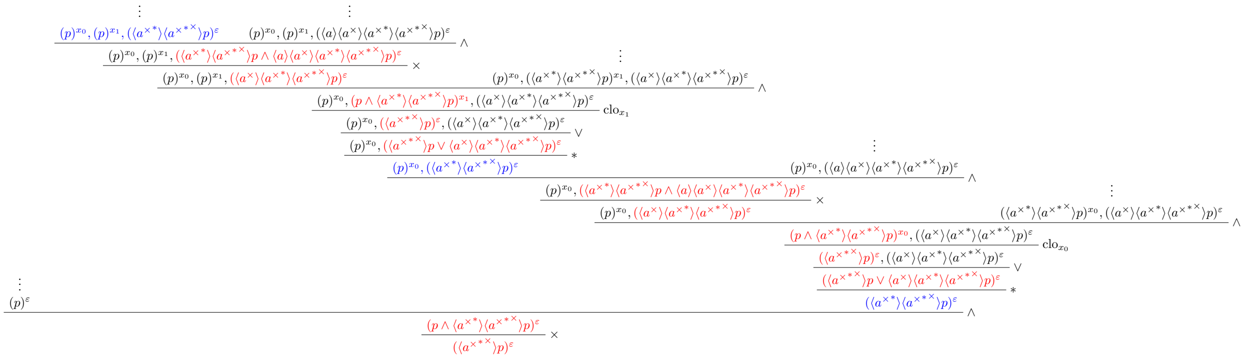


Figure 16: a cycle detection fail for the sequent $\langle a^{\times * \times}\rangle p^{\varepsilon}$, where the sequents marked in blue are the sequents for which an undesired cycle should be detected

If Angel has a strategy to ensure p holds after either playing game a a certain number of times, or after playing b a certain number of times ($\langle a^* \sqcup b^*\rangle p$), then Angel also has a strategy to ensure p holds after playing the game where Angel can choose to play either a or b a certain number of times ($\langle (a \sqcup b)^*\rangle p$). $\langle a^* \sqcup b^*\rangle p \to \langle (a \sqcup b)^*\rangle p$ must therefore be valid, and we can try the sequent $(\langle a^{d^\times} \sqcap b^{d^\times}\rangle \neg p \vee \langle (a \sqcup b)^*\rangle p)^{\varepsilon}$. For the Procrastination or Pre-Emptive mod_m Strategy, we obtain the same proof. Supposedly, there is only a difference between the resulting proofs for these two strategies if there are two or more nested least fixpoint formulae. If we use the Simple clo or Greedy Strategy, we obtain a slightly smaller proof. Both proofs are shown in figure 17. Once again, they can also be found in the table in appendix B.1, and in more detail in appendix B.2.

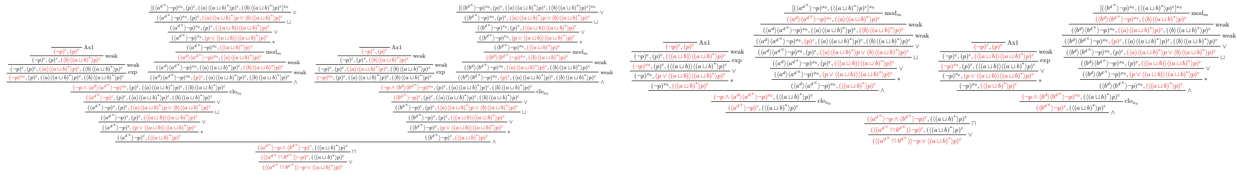


Figure 17: proofs for the sequent $(\langle a^{d^\times} \sqcap b^{d^\times}\rangle \neg p \vee \langle (a \sqcup b)^*\rangle p)^{\varepsilon}$, on the left for the Procrastination or Pre-Emptive mod_m Strategy, containing 37 rule applications, and on the right for the Simple clo or Greedy Strategy, containing 31 rule applications.

The next proof relies on the fact that Angel having a strategy to ensure p holds after playing a game, where they can choose whether to play a or b, a certain number of times ($\langle (a \sqcup b)^*\rangle p$), is essentially the same as having a strategy to ensure p holds after playing a game, where they have to play a a number of times, then b a number of times, a certain number of times ($\langle (a^* \,;\, b^*)^*\rangle p$). Thus, since $\langle (a \sqcup b)^*\rangle p \to \langle (a^* \,;\, b^*)^*\rangle p$ holds, the sequent $(\langle (a^d \sqcap b^d)^\times\rangle \neg p \vee \langle (a^* \,;\, b^*)^*\rangle p)^{\varepsilon}$ must be valid. In figure 18, we show the resulting proof for the Pre-Emptive mod_m Strategy. Once again, it is slightly smaller than the variant for the Procrastination Strategy. For details, check appendix B.2.

If instead, we try to apply the clo rule as early as possible (we try the Simple clo Strategy or the Greedy Strategy), the resulting proof gets too large to display properly, thus we show the general shape and size of it in figures 19 and 20. We include the versions for both the Simple clo Strategy, and the Greedy Strategy. Both have the same shape, but the latter has less than half as many nodes.

The last two proofs to show are the proofs for the sequents used in example proofs in [22].

For the proofs for the sequent $(\langle (a^d \sqcup b^d)^*\rangle \neg p \vee \langle (a^* \,;\, b)^\times\rangle p)^{\varepsilon}$, no new insights are gained, thus we do not show them here, but they can be found in appendix B.1. For the Simple clo and Greedy Strategies, we obtain a slightly smaller proof again, and since this proof does not contain a nested least fixpoint formula, the ordering of the mod_m rule does not change the proof.

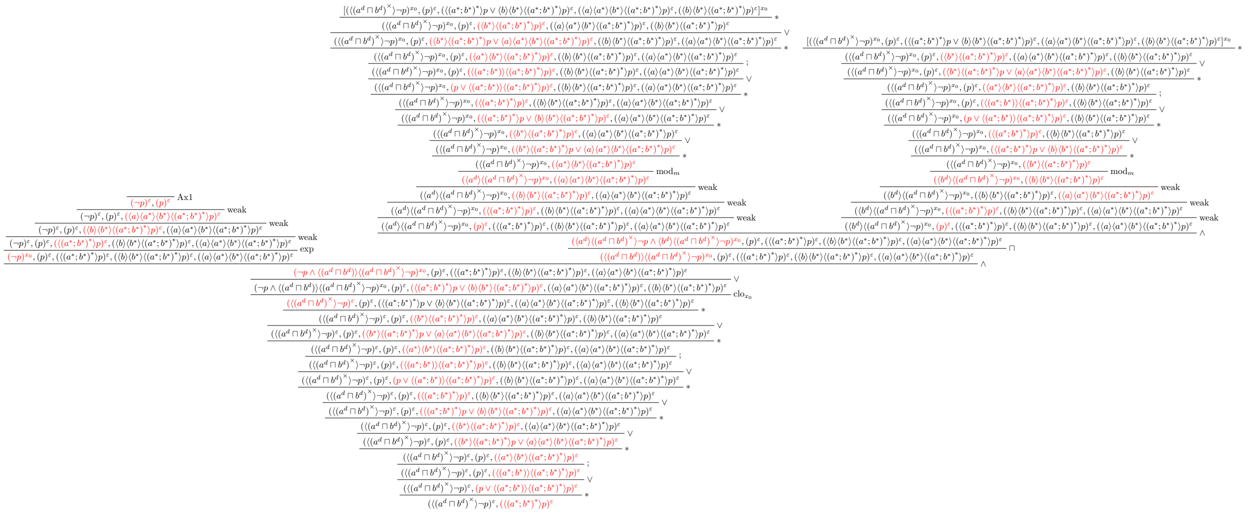


Figure 18: proof for $(\langle (a^d \sqcap b^d)^\times \rangle \neg p \vee \langle (a^* ; b^*)^* \rangle p)^\varepsilon$, using the Pre-Emptive mod_m Strategy; contains 49 rule applications

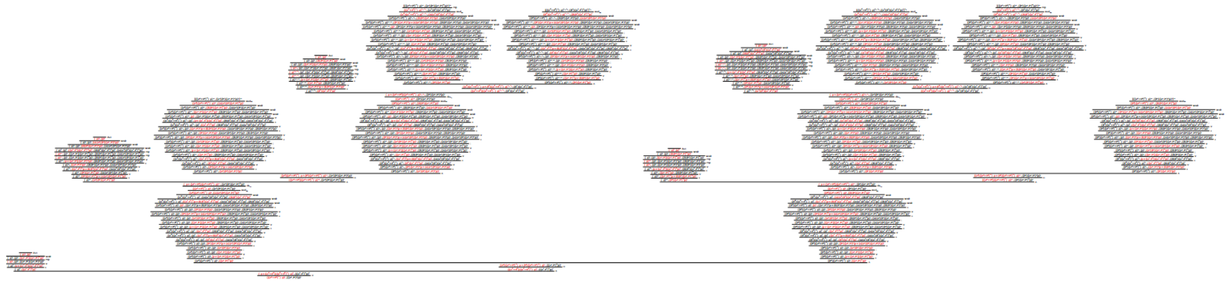


Figure 19: monster proof for the sequent $\langle a^{d^\times} \sqcap b^{d^\times} \rangle \neg p \vee \langle (a^* ; b^*)^* \rangle p^\varepsilon$, using the Simple clo Strategy

Figure 20: monster proof for the sequent $\langle a^{d^\times} \sqcap b^{d^\times} \rangle \neg p \vee \langle (a^* ; b^*)^* \rangle p^\varepsilon$, using the Greedy Strategy

The proof obtained from the Simple clo or Greedy Strategy is similar to the example proof used in [22]. The depth of the proof is essentially the same, the same rules are applied, only in a different order.

The final proof we try to find is the proof for the sequent $(\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q \vee \langle ((p? ; a^\times) \sqcup (\neg p? ; b^\times))^\times \rangle q)^\varepsilon$.

Figures 21, 22, 23, and 24 show the proofs found for this sequent, using the Procrastination, Pre-Emptive mod_m, Simple clo, and Greedy Strategies, respectively.

Something to note besides the size of the proofs, is the fact that the first two took just several minutes to find, considerably longer than any other proof search so far.

The proof found using the Greedy Strategy is the smallest of the found proofs, but is still not nearly as short as the example proof in [22], although it follows a similar structure. This example proof is shown in

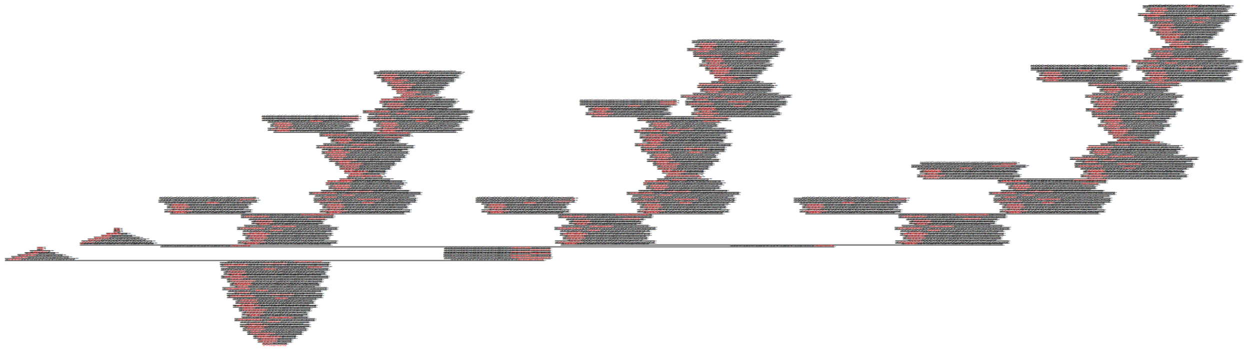


Figure 21: monster proof for the sequent $(\langle(a^{d^*} \sqcup b^{d^*})^*\rangle\neg q \vee \langle((p?\,; a^\times) \sqcup (\neg p?\,; b^\times))^\times\rangle q)^\varepsilon$, for the Procrastination Strategy. It took over 4 minutes on average to find the proof, and it contains 442 rule applications

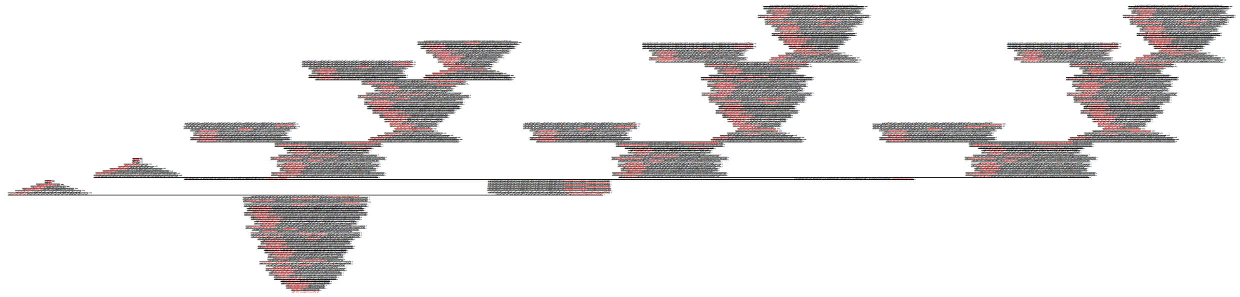


Figure 22: monster proof for the sequent $(\langle(a^{d^*} \sqcup b^{d^*})^*\rangle\neg q \vee \langle((p?\,; a^\times) \sqcup (\neg p?\,; b^\times))^\times\rangle q)^\varepsilon$, for the Pre-Emptive mod_m Strategy. It took over 3.5 minutes on average to find the proof, and it contains 332 rule applications

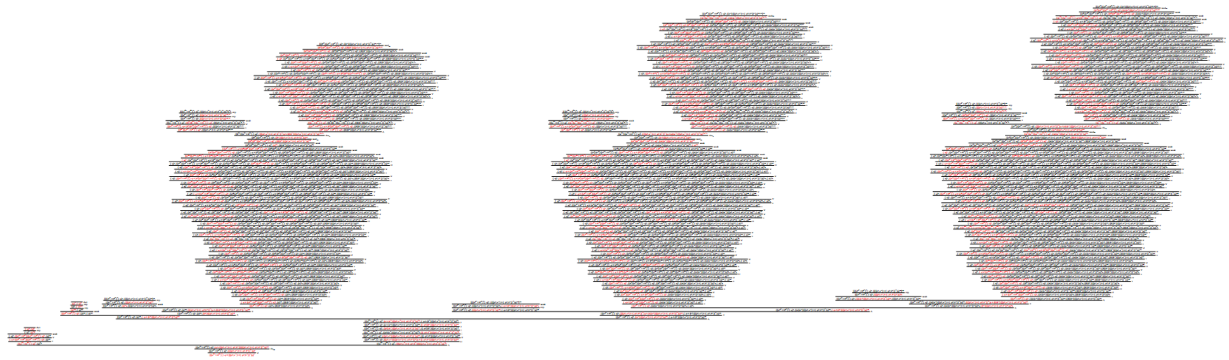


Figure 23: monster proof for the sequent $(\langle(a^{d^*} \sqcup b^{d^*})^*\rangle\neg q \vee \langle((p?\,; a^\times) \sqcup (\neg p?\,; b^\times))^\times\rangle q)^\varepsilon$, for the Simple clo Strategy. It took just over a second find the proof, but it still contains 274 rule applications

figure 25 for reference.

The main difference of the found proofs to the example proof, is the fact that each of these found proofs has a × rule application in between two clo rule applications on each branch of the proof. This used to be a clo rule application, but as no assumption for the rule was found, clo was replaced by × in post-processing. Each of these proofs thus contains an unnecessary greatest fixpoint formula unfolding, which results in unnecessary branching. And furthermore, there are many unnecessary least fixpoint formula unfoldings present, especially in the proofs with a late mod_m rule application.

Figure 24: monster proof for the sequent $(\langle(a^{d^*} \sqcup b^{d^*})^*\rangle\neg q \vee \langle((p? \,; a^\times) \sqcup (\neg p? \,; b^\times))^\times\rangle q)^\varepsilon$, for the Greedy Strategy. It took just over a second find the proof, but it still contains 107 rule applications

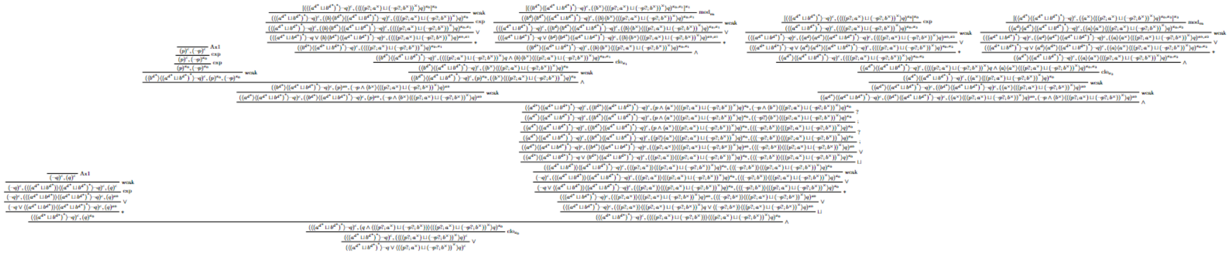


Figure 25: proof for the sequent $(\langle(a^{d^*} \sqcup b^{d^*})^*\rangle\neg q \vee \langle((p? \,; a^\times) \sqcup (\neg p? \,; b^\times))^\times\rangle q)^\varepsilon$ that was used as an example in [22] and contains 49 rule applications

8 Conclusion

8.1 Evaluation

The research question posed in section 1.2 was “How can we carry out Automated Theorem Proving with a practical implementation of the CloG proof system?”

To answer this question, we have developed several strategies for proof search (see section 5) and implemented them, first in pseudocode (see section 6.2), and then in Rascal (see appendix C). For all the sequents that have been tested, proofs have been found with all the strategies. It remains to be seen which of these strategies is the best, or how they can be approved further, as there appears to be a trade-off between execution time and number of clo rule applications.

In section 4, requirements have been set out for this tool. Most of these have been satisfied:

Requirements R-1.1 to R-1.4 are met: there is a way for a user to input a sequent, namely in a `.seq` file, the tool can apply automated proof search to this inputted sequent, through a proof search algorithm, and the program can output the proof if a proof is found, or indicate if one could not be found. For soundness of the tool (R-1.5), a full formal soundness proof remains in order, but the tool should be sound, as the requirements and additional side conditions for each of the rules have been implemented. As for completeness (R-2.1) of the tool, this is yet to be proven, and is further discussed in section 8.2. A user being able to specify the rule order (R-2.2) was unfortunately not implemented. The user would have to swap around the `tryApply<Rule>()` functions in the main `proofSearch()` function itself. Indicating active sequents (R-3.1) and proof search strategies (R-3.2) have been implemented, whereas live user-interaction (R-3.3), and a user-friendly UI (R-3.4) have not. For the non-functional requirements, these are met mostly by the unit/integration tests, documentation, and code organization.

8.2 Future Work

As for future work on the topic, there are a number of things that can be expanded on and improved upon for this tool.

Firstly, some formal analysis could be done on the tool. Can we formally prove that the strategies discussed in 5 are complete? We have explained the exchangeablility of most of the rule applications, and even argued how the clo rule can be "moved up" in a proof with respect to simple rules. This implies that a proof search where only a single ordering of rule applications is possible (save for the mod_m rule), if we try to apply the clo rule last. The completeness for the Simple clo and Greedy Strategies might be even harder to prove, as it relies on the assumption that the clo rule can also be "moved down" with respect to simple rules. We have provided no full formal proof for completeness for these strategies, as that fell outside of the scope of this project.

There was also another implicit design decision, which relates to the weak and exp rule ordering discussed in section 5.2.2. While we can exchange the weak and exp rule applications with any of the simple rules, we might not be able to exchange it with the clo rule application, as that would once again entail "moving down" the clo rule application. One could imagine having to weaken a sequent before a clo rule application, to limit the number of side-formulae, so that the assumption further up in the proof would be easier to match. However, we could not think of any examples where this might be necessary. In most, if not all cases, the side formulae necessary can either be retained, or regenerated to match the clo rule assumption.

A final thing that might stand in the way of completeness is the fact that we do not try to apply the × rule in any of the proof search algorithms. Because when we did, it caused problems, both because of the fact that it is not exchangeable with the clo rule, and because leaving it in would allow for specific cases of infinite recursion (as illustrated by the example in section 7.2.11). As I could not come up with any scenario in which a × rule application is *necessary* for a proof, I would conjecture that this rule is not needed for the completeness of the tool, and leaving it out does not make the proof search incomplete. Even the × rules converted from clo rules without discharged assumptions might not be necessary, as we can often derive a proof where we could find an assumption for that closure rule, as we argued for sequent 26 and 27 in section 7.2.9.

Besides soundness and completeness, there might be ways to improve the efficiency of the proof search. There are probably better ways to handle the cycle detection. A possible strategy for cycle detection to explore would be the following:

Like before, we save previous sequents in a list, and we detect a potentially desired cycle if our current sequent is a superset of the saved sequent, and has a greatest fixpoint formula with one more name than the same formula in the saved sequent. If such a cycle is detected for the current sequent, we look for the position of the saved sequent, and we look along the path or *trace* from the saved sequent to the current sequent. If there is a clo rule application on this trace applied to the greatest fixpoint formula mentioned before, we know we can reach the assumption for this clo rule application from the current sequent. All we have to do is apply weak and exp rules to the current sequent to match the saved sequent (apart from that one extra name in the saved fixpoint formula), and then apply all the rules between the saved sequent and the clo rule application to the current sequent as well. After this, we have reached the assumption for the clo rule, and we can discharge it.

Another potential strategy, which might result in smaller proofs, based on findings in section 7.2.9, would be to try to apply the clo rule as early as possible at the start of a proof, but as late as possible either after a mod_m application, or more generally whenever we are looking to find the assumption for an already used clo rule. However, we would need a more general strategy if there are multiple greatest fixpoints to unfold. A strategy like this would likely involve some kind of directed proof search, in order to match each clo rule's assumption.

There might also be better ways to apply the weak and exp rules without increasing the time complexity or search space of the proof search, so that we can obtain proofs like the last one for sequent 29. Or alternatively, we could take the proofs resulting from this tool, and do additional post-processing, where we go through the proof and skip or reorder rule applications in order to make the proof as small, efficient, or easily readable as possible. A final possibility to potentially achieve smaller proofs would be to implement some sort of breadth-first or iterative deepening search.

9 Acknowledgements

I would like to thank this project's supervisor, Prof. Dr. H. H. Hansen, for her help in finding sources, answering questions, and understanding much of the underlying theory. And for our weekly meetings discussing the state of the project and discussing various technical details of the CloG proof system, ideas for proof strategies and more.

I would also like to thank Christopher Worthington for allowing me to extend the Rascal code he used for his thesis last year [22].

Furthermore, my second supervisor, Dr. D.R.S. (Revantha) Ramanayake also helped a lot in my high-level understanding of proof search strategies within sequent systems and how I might apply this in CloG.

Finally, I would like to thank Prof. Dr. T. (Tijs) van der Storm for helping me understand the Rascal language and giving me advice on the usage of Rascal for Automated Theorem Proving.

References

[1] Bahareh Afshari and Graham Emil Leigh. Cut-free completeness for modal mu-calculus. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, LICS '17, pages 1–12, Reykjavík, Iceland, June 2017. IEEE Press. ISBN 9781509030187. doi: 10.1109/LICS.2017.8005088.

[2] Julian Bradfield and Colin Stirling. Modal mu-calculi. In Patrick Blackburn, Johan Van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*, chapter 12, pages 721–756. Elsevier, 2007. doi: 10.1016/S1570-2464(07)80015-2. URL `https://www.sciencedirect.com/science/article/pii/S1570246407800152`.

[3] James Brotherston. Cyclic proofs for first-order logic with inductive definitions. In *TABLEAUX'05*, volume 3702, pages 78–92. Springer-Verlag, 2005.

[4] James Brotherston. *Sequent Calculus Proof Systems for Inductive Definitions.* PhD thesis, University of Edinburgh, November 2006.

[5] James Brotherston, Nikos Gorogiannis, and Rasmus L. Petersen. A generic cyclic theorem prover. In Ranjit Jhala and Atsushi Igarashi, editors, *Programming Languages and Systems*, pages 350–367, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 9783642351822. doi: 10.1007/978-3-642-35182-2_25.

[6] Dai Clegg and Richard Barker. *Case Method Fast-Track: A Rad Approach.* Addison-Wesley Longman Publishing Co., Inc., USA, 1994. ISBN 020162432X.

[7] Sebastian Enqvist, Helle Hvid Hansen, Clemens Kupke, Johannes Marti, and Yde Venema. Completeness for game logic. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. Institute of Electrical and Electronics Engineers (IEEE), 2019. doi: 10.1109/LICS.2019.8785676.

[8] Gerhard Gentzen. Investigations into logical deduction. *American Philosophical Quarterly*, 1(4):288–306, 1964. ISSN 00030481. URL `http://www.jstor.org/stable/20009142`.

[9] Tobias Gleißner, Alexander Steen, and Christoph Benzüller. Theorem provers for every normal modal logic. In Thomas Eiter and David Sands, editors, *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46 of *EPiC Series in Computing*, pages 14–30. EasyChair, 2017. doi: 10.29007/jsb9. URL `https://easychair.org/publications/paper/6bjv`.

[10] Emile Hazard and Denis Kuperberg. Cyclic proofs for transfinite expressions. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*, volume 216 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 9783959772181. doi: 10.4230/LIPIcs.CSL.2022.23. URL `https://drops.dagstuhl.de/opus/volltexte/2022/15743`.

[11] Paul Klint, Tijs van der Storm, and Jurgen Vinju. Rascal: A domain specific language for source code analysis and manipulation. In *2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 168–177, 2009. doi: 10.1109/SCAM.2009.28.

[12] Donald Mackenzie. The automation of proof: A historical and sociological exploration. *IEEE Annals of the History of Computing*, 17(3):7–29, 1995. doi: 10.1109/85.397057.

[13] Talal Maghrabi. Automated theorem proving: An overview. *The Arabian Journal for Science and Engineering*, 22(2B):245–258, October 1997.

[14] Sara Negri, Jan von Plato, and Aarne Ranta. *Structural Proof Theory*, chapter 1, pages 1–24. Cambridge University Press, 2001. doi: 10.1017/CBO9780511527340.

[15] Nicola Olivetti, Gian Luca Pozzato, and Camilla B. Schwind. A sequent calculus and a theorem prover for standard conditional logics. *ACM Transactions on Computational Logic*, 8(4):22—-73, August 2007. ISSN 1529-3785. doi: 10.1145/1276920.1276924. URL `https://arxiv.org/abs/cs/0407064`.

[16] Eric Pacuit. *Neighborhood Semantics for Modal Logic*, chapter 1, pages 5–39. Cham, Switzerland: Springer, 2017.

[17] Rohit Parikh. The logic of games and its applications. *Annals of Discrete Mathematics*, 24, December 1985. doi: 10.1016/S0304-0208(08)73078-0.

[18] Marc Pauly. *Logic for Social Software.* PhD thesis, Universiteit van Amsterdam, January 2001.

[19] Marc Pauly and Rohit Parikh. Game logic - an overview. *Studia Logica*, 75(2):165–182, 2003. doi: 10.1023/A:1027354826364.

[20] William Poundstone. *Prisoner's dilemma.* Doubleday, New York, 1992. ISBN 9780385415804.

[21] Andrea Aler Tubella and Lutz Straßburger. Introduction to deep inference. Lecture, August 2019. URL `https://hal.inria.fr/hal-02390267`.

[22] Chris Worthington. *Proof Transformations for Game Logic.* Bachelor's thesis, computing science, Rijksuniversiteit Groningen, 2021.

[23] Richard Zach. *Boxes and Diamonds: An Open Introduction to Modal Logic*, chapter 1, pages 1–20. Open Logic Project, 2019.

A Implementation Changes w.r.t. Pseudocode

There are some minor differences between the pseudocode described in section 6.2, and the actual code found in appendix C. These changes do not have to do with the functionality of the proof search algorithm itself, but are changed for the sake of a clearer code organization and easier unit testing.

A.1 Data Types

`CloGSequent`s are implemented as lists of terms, rather than sets, so that we can refer to a specific term within the sequent. Because we use lists, we remove duplicate terms from the current sequent at the start of the `proofSearch()` function.

A.2 Maximum Depth

In the Rascal code, it is possible to indicate a maximum depth for the proof search. This depth is given as a 4th parameter to the `proofSearch()` function. This depth is then decreased by 1 for each recursive call to the `proofSearch()` function. The function returns `noProof()` if the depth reaches 0. For a non-depth-limited proof search, we can just set a depth of -1, which should find a proof if it is no deeper than the maximum integer size (and a stack overflow error would be given before that point).

This maximum depth was mostly used for manual testing, to prevent the program from getting stuck in infinite recursion. Also, if we return `proof(CloGLeaf())` rather than `noProof()` when the depth reaches 0, we obtain a partial proof, and might be able to tell what the program might be getting stuck on.

A.3 TryApply<Rule> vs Apply<Rule>

In the Rascal code, each of the `tryApply<Rule>()` functions for which the pseudocode is given in section 6.2.5) is split up into two separate functions. If we take for instance the $\vee$ rule, we have a `tryApplyOr()` function, which attempts to apply the $\vee$ rule to each of the terms in the current sequent, and if it can be successfully applied, it calls `proofSearch()` on the resulting sequent. The actual function responsible for checking whether the $\vee$ rule can be applied to a specific term and applying it if possible, is `applyOr()`. It can return the resulting sequent, or `noSeq()` if the $\vee$ rule could not be applied. This works similarly for the other rules. The side conditions for the $*$ and clo rules are also checked within the `applyIter()` and `applyClo()` functions respectively, rather than the `tryApply<Rule>()` variant. Additionally to the algorithms described in section 6.2.2, these rule application functions also keep track of the current depth (and decrement it upon calling `proofSearch()`), and mark terms to which a rule is successfully applied as active.

B Testing Sequents and Proofs

B.1 Tabular Overview

The following table shows the 39 tested sequents and the proofs found for each applied proof strategy.

The first column indicates the number of the sequent. The second column contains the sequent itself. The third column indicates the proof strategy or strategies that were applied to the sequent, in order to produce the proof in the 4th column. For these proof strategies, 1 stands for the Procrastination Strategy (definition 5.3), 2 for the Pre-Emptive mod_m Strategy (definition 5.4), 3 for the Simple clo strategy (definition 5.5), and 4 for the Greedy Strategy (definition 5.6).

Seq Nr	**Sequent**	**Strategy**	**Proof**
1	(empty sequent)	Any	No Proof
2	p^{ε}	Any	No Proof
3	p^{x_0}	Any	No Proof
4	$p^{\varepsilon}, q^{\varepsilon}$	Any	No Proof

5	p^{x_0,x_1}, q^{x_1,x_2}	Any	No Proof
6	$p^{\varepsilon}, \neg p^{\varepsilon}$	Any	$\dfrac{}{(p)^{\varepsilon}, (\neg p)^{\varepsilon}}\,\text{Ax1}$
7	$p^{x_0,x_1}, \neg p^{x_0,x_1}$	Any	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(p)^{\varepsilon}, (\neg p)^{\varepsilon}}\,\text{Ax1}}{(p)^{\varepsilon}, (\neg p)^{x_2}}\,\text{exp}}{(p)^{\varepsilon}, (\neg p)^{x_1,x_2}}\,\text{exp}}{(p)^{x_1}, (\neg p)^{x_1,x_2}}\,\text{exp}}{(p)^{x_0,x_1}, (\neg p)^{x_1,x_2}}\,\text{exp}$
8	$p^{\varepsilon}, \neg q^{\varepsilon}, r^{\varepsilon}, q^{\varepsilon}$	Any	$\dfrac{\dfrac{\dfrac{}{(\neg q)^{\varepsilon}, (q)^{\varepsilon}}\,\text{Ax1}}{(\neg q)^{\varepsilon}, (r)^{\varepsilon}, (q)^{\varepsilon}}\,\text{weak}}{(p)^{\varepsilon}, (\neg q)^{\varepsilon}, (r)^{\varepsilon}, (q)^{\varepsilon}}\,\text{weak}$
9	$p^{\varepsilon}, \neg q^{x_1,x_2}, r^{x_1}, q^{x_0}$	Any	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg q)^{\varepsilon}, (q)^{\varepsilon}}\,\text{Ax1}}{(\neg q)^{\varepsilon}, (q)^{x_0}}\,\text{exp}}{(\neg q)^{\varepsilon}, (r)^{x_1}, (q)^{x_0}}\,\text{weak}}{(\neg q)^{x_2}, (r)^{x_1}, (q)^{x_0}}\,\text{exp}}{(\neg q)^{x_1,x_2}, (r)^{x_1}, (q)^{x_0}}\,\text{exp}}{(p)^{\varepsilon}, (\neg q)^{x_1,x_2}, (r)^{x_1}, (q)^{x_0}}\,\text{weak}$
10	$\langle a \rangle p^{x_0} *$	Any	$[(\langle a^{\times} \rangle p)^{x_0}]^{x_0}$
11	$(p \vee \neg p)^{\varepsilon}$	Any	$\dfrac{\dfrac{}{(p)^{\varepsilon}, (\neg p)^{\varepsilon}}\,\text{Ax1}}{(p \vee \neg p)^{\varepsilon}}\,\vee$
12	$p^{x_0}, (q \vee \neg q)^{x_0,x_1}, r^{x_2}$	Any	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(q)^{\varepsilon}, (\neg q)^{\varepsilon}}\,\text{Ax1}}{(q)^{\varepsilon}, (\neg q)^{\varepsilon}, (r)^{x_2}}\,\text{weak}}{(q)^{\varepsilon}, (\neg q)^{x_1}, (r)^{x_2}}\,\text{exp}}{(q)^{\varepsilon}, (\neg q)^{x_0,x_1}, (r)^{x_2}}\,\text{exp}}{(q)^{x_1}, (\neg q)^{x_0,x_1}, (r)^{x_2}}\,\text{exp}}{(q)^{x_0,x_1}, (\neg q)^{x_0,x_1}, (r)^{x_2}}\,\text{exp}}{(p)^{x_0}, (q)^{x_0,x_1}, (\neg q)^{x_0,x_1}, (r)^{x_2}}\,\text{weak}}{(p)^{x_0}, (q \vee \neg q)^{x_0,x_1}, (r)^{x_2}}\,\vee$

13	$(p\wedge q)^\varepsilon, \neg p^\varepsilon, \neg q^\varepsilon$	Any	$\dfrac{\dfrac{\dfrac{}{(p)^\varepsilon,(\neg p)^\varepsilon}\,\text{Ax1}}{(p)^\varepsilon,(\neg p)^\varepsilon,(\neg q)^\varepsilon}\,\text{weak} \qquad \dfrac{\dfrac{}{(q)^\varepsilon,(\neg q)^\varepsilon}\,\text{Ax1}}{(q)^\varepsilon,(\neg p)^\varepsilon,(\neg q)^\varepsilon}\,\text{weak}}{(p\wedge q)^\varepsilon,(\neg p)^\varepsilon,(\neg q)^\varepsilon}\,\wedge$
14	$(\langle a\rangle p\vee\langle a^d\rangle\neg p)^\varepsilon$	Any	$\dfrac{\dfrac{\dfrac{}{(p)^\varepsilon,(\neg p)^\varepsilon}\,\text{Ax1}}{(\langle a\rangle p)^\varepsilon,(\langle a^d\rangle\neg p)^\varepsilon}\,\text{mod}_m}{(\langle a\rangle p\vee\langle a^d\rangle\neg p)^\varepsilon}\,\vee$
15	$(\langle a\sqcup b\rangle p\vee\langle a^d\rangle\neg p)^\varepsilon$	Any	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(p)^\varepsilon,(\neg p)^\varepsilon}\,\text{Ax1}}{(\langle a\rangle p)^\varepsilon,(\langle a^d\rangle\neg p)^\varepsilon}\,\text{mod}_m}{(\langle a\rangle p)^\varepsilon,(\langle b\rangle p)^\varepsilon,(\langle a^d\rangle\neg p)^\varepsilon}\,\text{weak}}{(\langle a\rangle p\vee\langle b\rangle p)^\varepsilon,(\langle a^d\rangle\neg p)^\varepsilon}\,\vee}{(\langle(a\sqcup b)\rangle p)^\varepsilon,(\langle a^d\rangle\neg p)^\varepsilon}\,\sqcup}{(\langle(a\sqcup b)\rangle p\vee\langle a^d\rangle\neg p)^\varepsilon}\,\vee$
16	$(\langle a\sqcap b\rangle p\vee\langle a^d\sqcup b^d\rangle\neg p)^\varepsilon$	Any	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(p)^\varepsilon,(\neg p)^\varepsilon}\,\text{Ax1}}{(\langle a\rangle p)^\varepsilon,(\langle a^d\rangle\neg p)^\varepsilon}\,\text{mod}_m}{(\langle a\rangle p)^\varepsilon,(\langle a^d\rangle\neg p)^\varepsilon,(\langle b^d\rangle\neg p)^\varepsilon}\,\text{weak} \qquad \dfrac{\dfrac{\dfrac{}{(p)^\varepsilon,(\neg p)^\varepsilon}\,\text{Ax1}}{(\langle b\rangle p)^\varepsilon,(\langle b^d\rangle\neg p)^\varepsilon}\,\text{mod}_m}{(\langle b\rangle p)^\varepsilon,(\langle a^d\rangle\neg p)^\varepsilon,(\langle b^d\rangle\neg p)^\varepsilon}\,\text{weak}}{(\langle a\rangle p\wedge\langle b\rangle p)^\varepsilon,(\langle a^d\rangle\neg p)^\varepsilon,(\langle b^d\rangle\neg p)^\varepsilon}\,\wedge}{(\langle a\rangle p\wedge\langle b\rangle p)^\varepsilon,(\langle a^d\rangle\neg p\vee\langle b^d\rangle\neg p)^\varepsilon}\,\vee}{(\langle(a\sqcap b)\rangle p)^\varepsilon,(\langle a^d\rangle\neg p\vee\langle b^d\rangle\neg p)^\varepsilon}\,\sqcap}{(\langle(a\sqcap b)\rangle p)^\varepsilon,(\langle(a^d\sqcup b^d)\rangle\neg p)^\varepsilon}\,\sqcup}{(\langle(a\sqcap b)\rangle p\vee\langle(a^d\sqcup b^d)\rangle\neg p)^\varepsilon}\,\vee$
17	$(\langle a\,;b\rangle p\vee\langle a^d\,;b^d\rangle\neg p)^\varepsilon$	Any	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(p)^\varepsilon,(\neg p)^\varepsilon}\,\text{Ax1}}{(\langle b\rangle p)^\varepsilon,(\langle b^d\rangle\neg p)^\varepsilon}\,\text{mod}_m}{(\langle a\rangle\langle b\rangle p)^\varepsilon,(\langle a^d\rangle\langle b^d\rangle\neg p)^\varepsilon}\,\text{mod}_m}{(\langle a\rangle\langle b\rangle p)^\varepsilon,(\langle(a^d;b^d)\rangle\neg p)^\varepsilon}\,;}{(\langle(a;b)\rangle p)^\varepsilon,(\langle(a^d;b^d)\rangle\neg p)^\varepsilon}\,;}{(\langle(a;b)\rangle p\vee\langle(a^d;b^d)\rangle\neg p)^\varepsilon}\,\vee$
18	$\langle p!\rangle\neg p^\varepsilon$	Any	$\dfrac{\dfrac{\dfrac{}{(p)^\varepsilon,(\neg p)^\varepsilon}\,\text{Ax1}}{(p\vee\neg p)^\varepsilon}\,\vee}{(\langle p!\rangle\neg p)^\varepsilon}\,!$

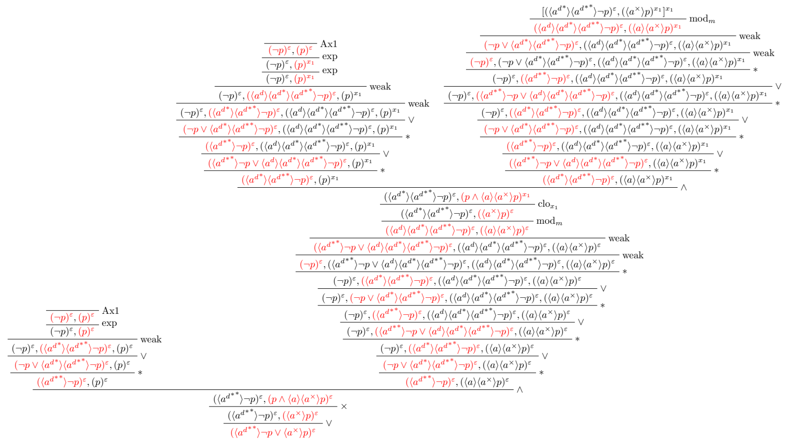
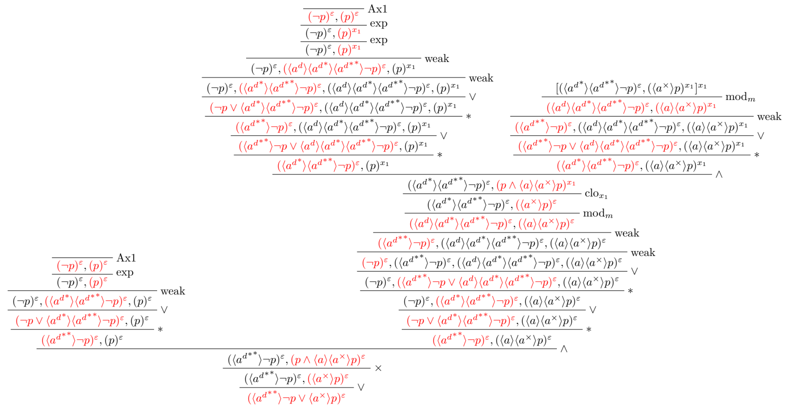
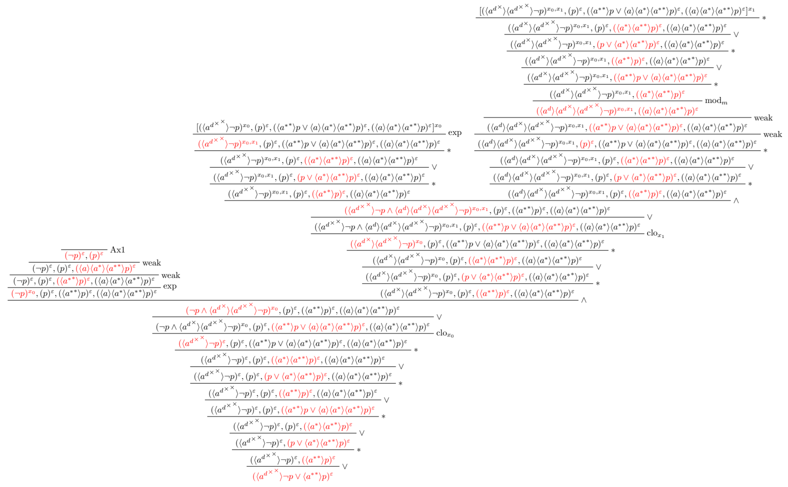
19	$\langle p?\rangle q^\varepsilon, \neg p^\varepsilon, \neg q^\varepsilon$	Any	$\dfrac{\dfrac{\dfrac{}{(p)^\varepsilon,(\neg p)^\varepsilon}\text{Ax1}}{(p)^\varepsilon,(\neg p)^\varepsilon,(\neg q)^\varepsilon}\text{weak}\qquad\dfrac{\dfrac{}{(q)^\varepsilon,(\neg q)^\varepsilon}\text{Ax1}}{(q)^\varepsilon,(\neg p)^\varepsilon,(\neg q)^\varepsilon}\text{weak}}{\dfrac{(p\wedge q)^\varepsilon,(\neg p)^\varepsilon,(\neg q)^\varepsilon}{(\langle p?\rangle q)^\varepsilon,(\neg p)^\varepsilon,(\neg q)^\varepsilon}?}\wedge$
20	$(\langle a^d\rangle\neg p\vee\langle a\rangle(p\vee q))^\varepsilon$	Any	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^\varepsilon,(p)^\varepsilon}\text{Ax1}}{(\neg p)^\varepsilon,(p)^\varepsilon,(q)^\varepsilon}\text{weak}}{(\neg p)^\varepsilon,(p\vee q)^\varepsilon}\vee}{(\langle a^d\rangle\neg p)^\varepsilon,(\langle a\rangle(p\vee q))^\varepsilon}\text{mod}_m}{(\langle a^d\rangle\neg p\vee\langle a\rangle(p\vee q))^\varepsilon}\vee$
21	$(\neg p\vee\langle a^*\rangle p)^\varepsilon$	Any	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^\varepsilon,(p)^\varepsilon}\text{Ax1}}{(\neg p)^\varepsilon,(p)^\varepsilon,(\langle a\rangle\langle a^*\rangle p)^\varepsilon}\text{weak}}{(\neg p)^\varepsilon,(p\vee\langle a\rangle\langle a^*\rangle p)^\varepsilon}\vee}{(\neg p)^\varepsilon,(\langle a^*\rangle p)^\varepsilon}*}{(\neg p\vee\langle a^*\rangle p)^\varepsilon}\vee$
22	$(\langle a^d\rangle\neg p\vee\langle a^*\rangle p)^\varepsilon$	Any	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^\varepsilon,(p)^\varepsilon}\text{Ax1}}{(\neg p)^\varepsilon,(p)^\varepsilon,(\langle a\rangle\langle a^*\rangle p)^\varepsilon}\text{weak}}{(\neg p)^\varepsilon,(p\vee\langle a\rangle\langle a^*\rangle p)^\varepsilon}\vee}{(\neg p)^\varepsilon,(\langle a^*\rangle p)^\varepsilon}*}{(\langle a^d\rangle\neg p)^\varepsilon,(\langle a\rangle\langle a^*\rangle p)^\varepsilon}\text{mod}_m}{(\langle a^d\rangle\neg p)^\varepsilon,(p)^\varepsilon,(\langle a\rangle\langle a^*\rangle p)^\varepsilon}\text{weak}}{(\langle a^d\rangle\neg p)^\varepsilon,(p\vee\langle a\rangle\langle a^*\rangle p)^\varepsilon}\vee}{(\langle a^d\rangle\neg p)^\varepsilon,(\langle a^*\rangle p)^\varepsilon}*}{(\langle a^d\rangle\neg p\vee\langle a^*\rangle p)^\varepsilon}\vee$

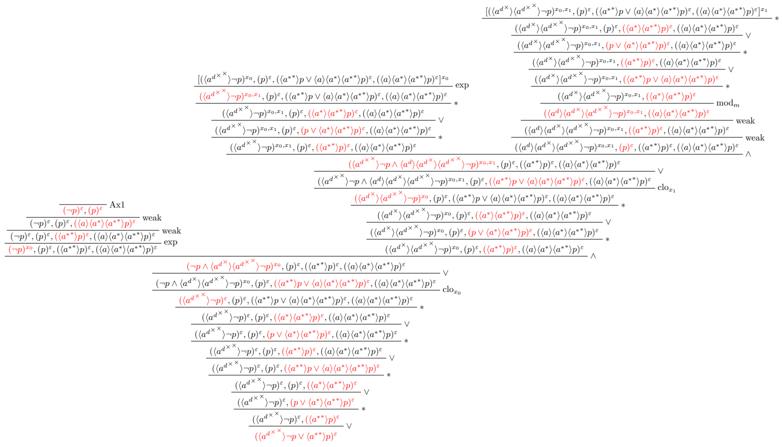
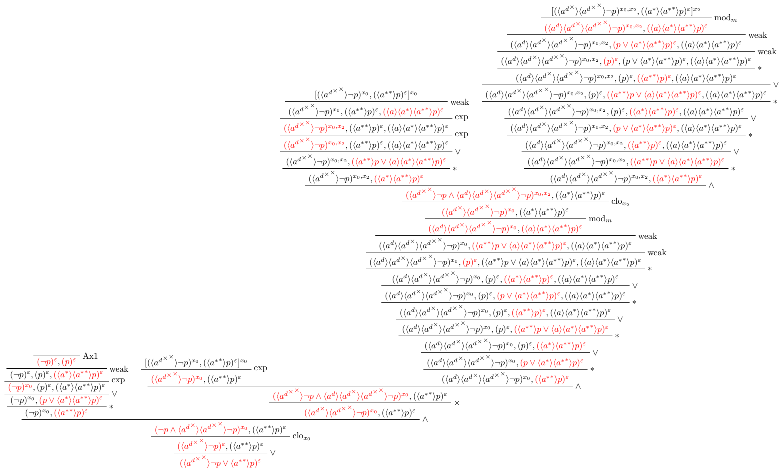
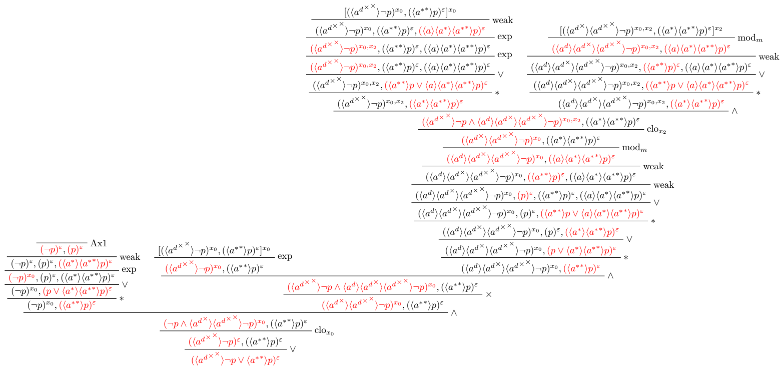
23	$(\langle a^d\rangle\neg p \vee \langle a^{**}\rangle p)^{\varepsilon}$	1, 3	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\,\text{Ax1}}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\text{weak}}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\text{weak}}{(\neg p)^{\varepsilon}, (p \vee \langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\vee}{(\neg p)^{\varepsilon}, (\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,*}{(\neg p)^{\varepsilon}, (\langle a^{**}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\vee}{(\neg p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,*}{(\langle a^d\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\text{mod}_m}{(\langle a^d\rangle\neg p)^{\varepsilon}, (\langle a^{**}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\text{weak}}{(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\text{weak}}{(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,*}{(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (p \vee \langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\vee}{(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,*}{(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\vee}{(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,*}{(\langle a^d\rangle\neg p)^{\varepsilon}, (p \vee \langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\vee}{(\langle a^d\rangle\neg p)^{\varepsilon}, (\langle a^{**}\rangle p)^{\varepsilon}}\,*}{(\langle a^d\rangle\neg p \vee \langle a^{**}\rangle p)^{\varepsilon}}\,\vee$
23	$(\langle a^d\rangle\neg p \vee \langle a^{**}\rangle p)^{\varepsilon}$	2, 4	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\,\text{Ax1}}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\text{weak}}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\text{weak}}{(\neg p)^{\varepsilon}, (p \vee \langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\vee}{(\neg p)^{\varepsilon}, (\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,*}{(\neg p)^{\varepsilon}, (\langle a^{**}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\vee}{(\neg p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,*}{(\langle a^d\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\text{mod}_m}{(\langle a^d\rangle\neg p)^{\varepsilon}, (\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\text{weak}}{(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\text{weak}}{(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\vee}{(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,*}{(\langle a^d\rangle\neg p)^{\varepsilon}, (p \vee \langle a^*\rangle\langle a^{**}\rangle p)^{\varepsilon}}\,\vee}{(\langle a^d\rangle\neg p)^{\varepsilon}, (\langle a^{**}\rangle p)^{\varepsilon}}\,*}{(\langle a^d\rangle\neg p \vee \langle a^{**}\rangle p)^{\varepsilon}}\,\vee$

24	$(\langle a^{d}\rangle\neg p \vee \langle a^{***}\rangle p)^{\varepsilon}$	1, 3	$\overline{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}$ Ax1 $\overline{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ weak $\overline{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ weak $\overline{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ weak $\overline{(\neg p)^{\varepsilon}, (p \vee \langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$ $\overline{(\neg p)^{\varepsilon}, (\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\neg p)^{\varepsilon}, (\langle a^{***}\rangle p \vee \langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$ $\overline{(\neg p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\neg p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p \vee \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$ $\overline{(\neg p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ mod_{m} $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (\langle a^{***}\rangle p \vee \langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ weak $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p \vee \langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ weak $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p \vee \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (p \vee \langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p \vee \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (p \vee \langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p \vee \langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (p \vee \langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p \vee \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p \vee \langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p \vee \langle a\rangle\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p \vee \langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (p \vee \langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p \vee \langle a^{*}\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (p \vee \langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$ $\overline{(\langle a^{d}\rangle\neg p)^{\varepsilon}, (\langle a^{***}\rangle p)^{\varepsilon}}$ $*$ $\overline{(\langle a^{d}\rangle\neg p \vee \langle a^{***}\rangle p)^{\varepsilon}}$ $\vee$

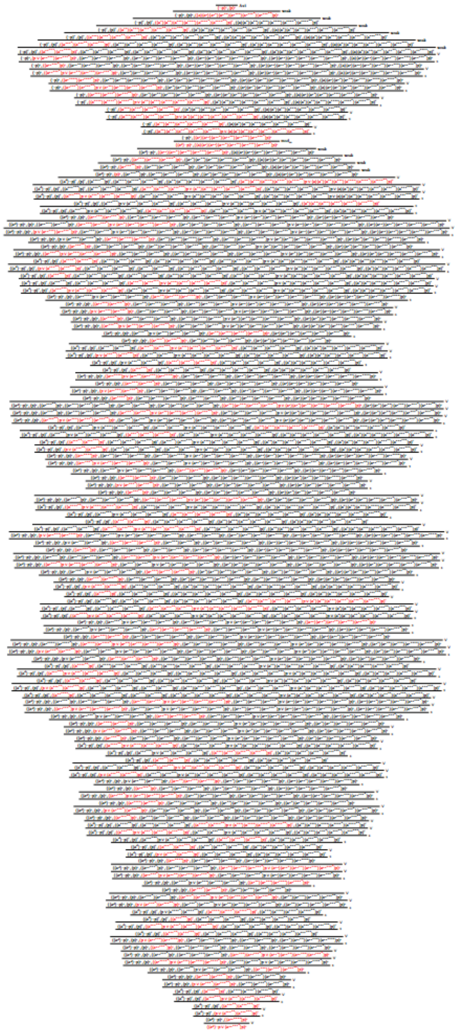
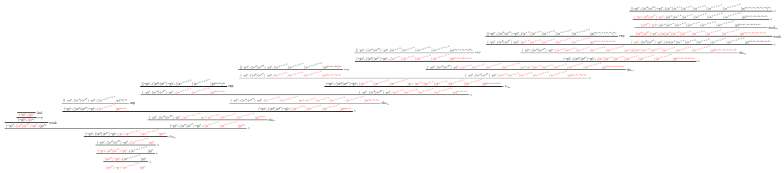
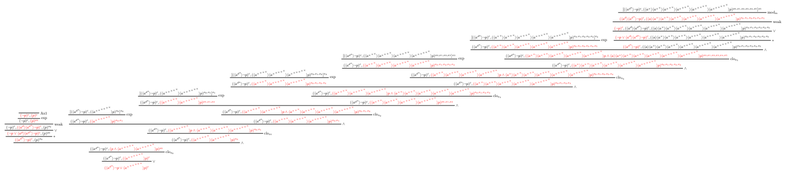
24	$(\langle a^d\rangle\neg p \vee \langle a^{***}\rangle p)^{\varepsilon}$	2, 4	$\overline{\qquad}$ Ax1 $(\neg p)^{\varepsilon}, (p)^{\varepsilon}$ $\overline{\qquad}$ weak $(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ weak $(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ weak $(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $\vee$ $(\neg p)^{\varepsilon}, (p \vee \langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $*$ $(\neg p)^{\varepsilon}, (\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $\vee$ $(\neg p)^{\varepsilon}, (\langle a^{***}\rangle p \vee \langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $*$ $(\neg p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $\vee$ $(\neg p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $*$ $(\neg p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ mod_m $(\langle a^d\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ weak $(\langle a^d\rangle\neg p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ weak $(\langle a^d\rangle\neg p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ weak $(\langle a^d\rangle\neg p)^{\varepsilon}, (\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ weak $(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $\vee$ $(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $\vee$ $(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p \vee \langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $*$ $(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p \vee \langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $*$ $(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $\vee$ $(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (p \vee \langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $*$ $(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p)^{\varepsilon}, (\langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $\vee$ $(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{***}\rangle p \vee \langle a^*\rangle\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $*$ $(\langle a^d\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $\vee$ $(\langle a^d\rangle\neg p)^{\varepsilon}, (p \vee \langle a^{**}\rangle\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $*$ $(\langle a^d\rangle\neg p)^{\varepsilon}, (\langle a^{***}\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $\vee$ $(\langle a^d\rangle\neg p \vee \langle a^{***}\rangle p)^{\varepsilon}$
25	$(\langle a^{d^{\times}}\rangle\neg p \vee \langle a^*\rangle p)^{\varepsilon}$	1, 2	$\overline{\qquad}$ Ax1 $(\neg p)^{\varepsilon}, (p)^{\varepsilon}$ $\overline{\qquad}$ weak $(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}$ $\overline{\qquad}$ exp $(\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}$ $[(\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}]^{x_0}$ $\overline{\qquad}$ $\vee$ $(\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (p \vee \langle a\rangle\langle a^*\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $*$ $(\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (\langle a^*\rangle p)^{\varepsilon}$ $\overline{\qquad}$ mod_m $(\langle a^d\rangle\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}$ $\overline{\qquad}$ weak $(\langle a^d\rangle\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}$ $\dfrac{(\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon} \qquad (\langle a^d\rangle\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}}{(\neg p \wedge \langle a^d\rangle\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}}\ \wedge$ $\overline{\qquad}$ clo_{x_0} $(\langle a^{d^{\times}}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $\vee$ $(\langle a^{d^{\times}}\rangle\neg p)^{\varepsilon}, (p \vee \langle a\rangle\langle a^*\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $*$ $(\langle a^{d^{\times}}\rangle\neg p)^{\varepsilon}, (\langle a^*\rangle p)^{\varepsilon}$ $\overline{\qquad}$ $\vee$ $(\langle a^{d^{\times}}\rangle\neg p \vee \langle a^*\rangle p)^{\varepsilon}$

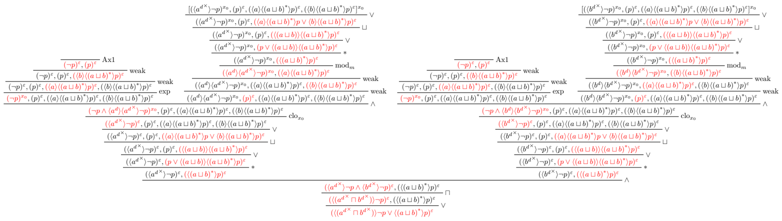
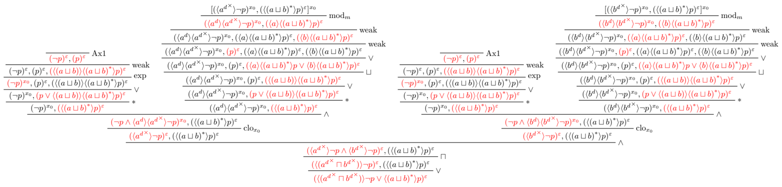
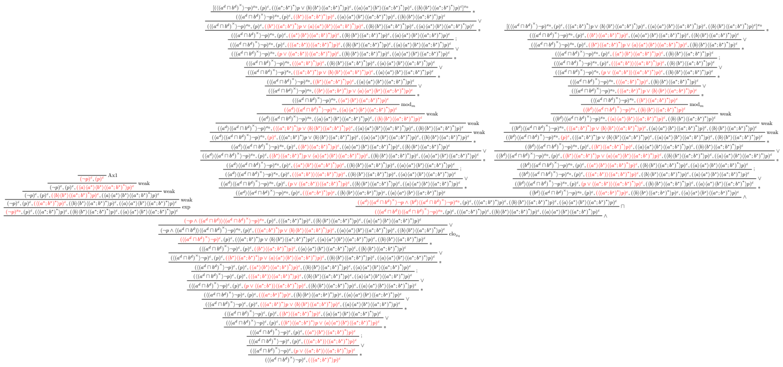
25	$(\langle a^{d^{\times}}\rangle\neg p \vee \langle a^{*}\rangle p)^{\varepsilon}$	3, 4	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\,\text{Ax1}}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle p)^{\varepsilon}}\,\text{weak}}{(\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle p)^{\varepsilon}}\,\text{exp}}{(\neg p)^{x_0}, (p \vee \langle a\rangle\langle a^{*}\rangle p)^{\varepsilon}}\,\vee}{(\neg p)^{x_0}, (\langle a^{*}\rangle p)^{\varepsilon}}\,* \qquad \dfrac{\dfrac{\dfrac{\dfrac{[(\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (\langle a^{*}\rangle p)^{\varepsilon}]^{x_0}}{(\langle a^{d}\rangle\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (\langle a\rangle\langle a^{*}\rangle p)^{\varepsilon}}\,\text{mod}_m}{(\langle a^{d}\rangle\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle a^{*}\rangle p)^{\varepsilon}}\,\text{weak}}{(\langle a^{d}\rangle\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (p \vee \langle a\rangle\langle a^{*}\rangle p)^{\varepsilon}}\,\vee}{(\langle a^{d}\rangle\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (\langle a^{*}\rangle p)^{\varepsilon}}\,*}{(\neg p \wedge \langle a^{d}\rangle\langle a^{d^{\times}}\rangle\neg p)^{x_0}, (\langle a^{*}\rangle p)^{\varepsilon}}\,\wedge}{(\langle a^{d^{\times}}\rangle\neg p)^{\varepsilon}, (\langle a^{*}\rangle p)^{\varepsilon}}\,\text{clo}_{x_0}}{(\langle a^{d^{\times}}\rangle\neg p \vee \langle a^{*}\rangle p)^{\varepsilon}}\,\vee$
26	$(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{\times}\rangle p)^{\varepsilon}$	1	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\,\text{Ax1}}{(\neg p)^{\varepsilon}, (p)^{x_0}}\,\text{exp}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (p)^{x_0}}\,\text{weak}}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (p)^{x_0}}\,\text{weak} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}]^{x_0}}{(\neg p)^{\varepsilon}, (\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}}\,*}{(\neg p \vee \langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}}\,\vee}{(\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}}\,*}{(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}}\,\vee}{(\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}}\,*}{(\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,\text{mod}_m}{(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,\text{weak}}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,\text{weak}}{(\neg p)^{\varepsilon}, (\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,*}{(\neg p)^{\varepsilon}, (\neg p \vee \langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,\vee}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,*}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (p \wedge \langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,\wedge}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (p \wedge \langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,\vee}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,\text{clo}_{x_0}}{(\neg p)^{\varepsilon}, (\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,*}{(\neg p)^{\varepsilon}, (\neg p \vee \langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,\vee}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,*}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,\vee}{(\neg p)^{\varepsilon}, (\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,*}{(\neg p \vee \langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,\vee}{(\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,*}{(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{\times}\rangle p)^{\varepsilon}}\,\vee$
26	$(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{\times}\rangle p)^{\varepsilon}$	2	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\,\text{Ax1}}{(\neg p)^{\varepsilon}, (p)^{x_0}}\,\text{exp}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (p)^{x_0}}\,\text{weak}}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (p)^{x_0}}\,\text{weak} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}]^{x_0}}{(\neg p)^{\varepsilon}, (\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}}\,*}{(\neg p \vee \langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}}\,\vee}{(\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}}\,*}{(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}}\,\vee}{(\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{x_0}}\,*}{(\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,\text{mod}_m}{(\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,\text{weak}}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,\text{weak}}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (p \wedge \langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,\wedge}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (p \wedge \langle a\rangle\langle a^{\times}\rangle p)^{x_0}}\,\vee}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,\text{clo}_{x_0}}{(\neg p)^{\varepsilon}, (\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,*}{(\neg p)^{\varepsilon}, (\neg p \vee \langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,\vee}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,*}{(\neg p)^{\varepsilon}, (\langle a^{d^{**}}\rangle\neg p \vee \langle a^{d}\rangle\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,\vee}{(\neg p)^{\varepsilon}, (\langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,*}{(\neg p \vee \langle a^{d^{*}}\rangle\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,\vee}{(\langle a^{d^{**}}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle p)^{\varepsilon}}\,*}{(\langle a^{d^{**}}\rangle\neg p \vee \langle a^{\times}\rangle p)^{\varepsilon}}\,\vee$

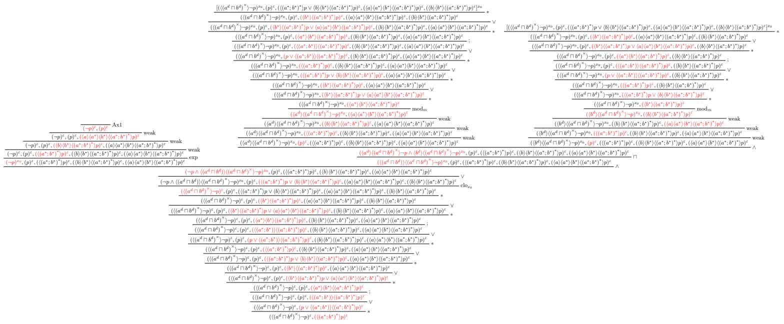
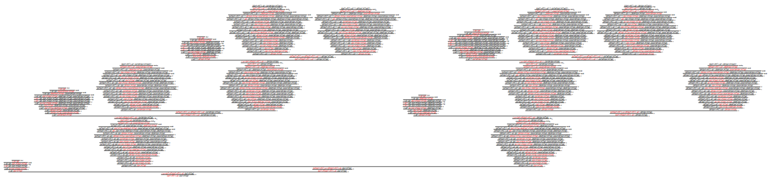
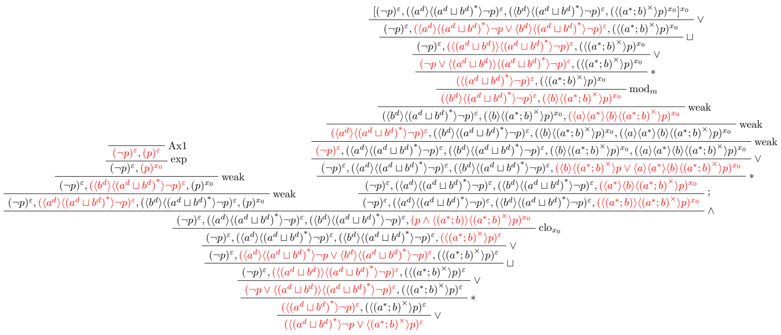
26	$(\langle a^{d^{**}}\rangle\neg p\vee\langle a^{\times}\rangle p)^{\varepsilon}$	3	
26	$(\langle a^{d^{**}}\rangle\neg p\vee\langle a^{\times}\rangle p)^{\varepsilon}$	4	
27	$(\langle a^{d^{\times\times}}\rangle\neg p\vee\langle a^{**}\rangle p)^{\varepsilon}$	1	

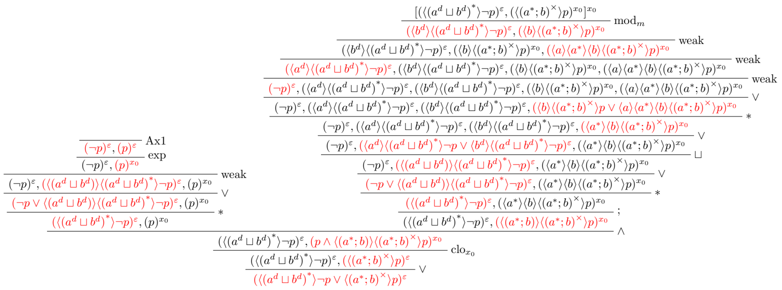
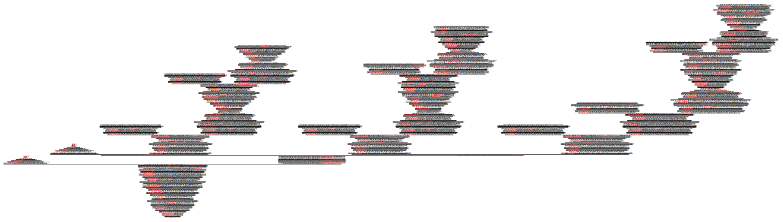
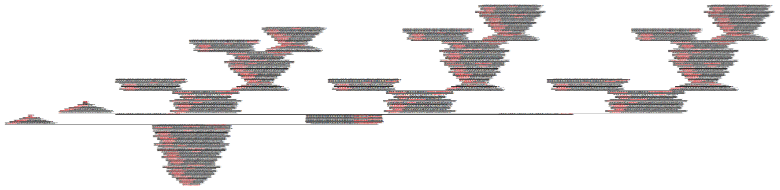
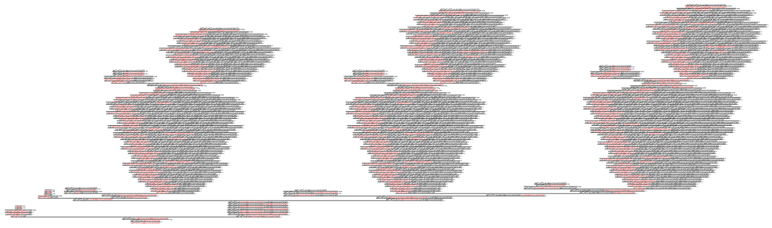
27	$(\langle a^{d^{\times\times}}\rangle\neg p \vee \langle a^{**}\rangle p)^{\varepsilon}$	2	
27	$(\langle a^{d^{\times\times}}\rangle\neg p \vee \langle a^{**}\rangle p)^{\varepsilon}$	3	
27	$(\langle a^{d^{\times\times}}\rangle\neg p \vee \langle a^{**}\rangle p)^{\varepsilon}$	4	

28	$(\neg p \vee \langle a^{******} \rangle p)^{\varepsilon}$	Any	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\,\mathrm{Ax1}}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^{*****} \rangle \langle a^{******} \rangle p)^{\varepsilon}}\,\mathrm{weak}}{(\neg p)^{\varepsilon}, (p \vee \langle a^{*****} \rangle \langle a^{******} \rangle p)^{\varepsilon}}\,\vee}{(\neg p)^{\varepsilon}, (\langle a^{******} \rangle p)^{\varepsilon}}\,*}{(\neg p \vee \langle a^{******} \rangle p)^{\varepsilon}}\,\vee$
29	$(\langle a^{d} \rangle \neg p \vee \langle a^{******} \rangle p)^{\varepsilon}$	1, 3	

29	$(\langle a^d\rangle\neg p \vee \langle a^{******}\rangle p)^\varepsilon$	2,4	
30	$(\langle a^{d^*}\rangle\neg p \vee \langle a^{\times^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}}\rangle p)^\varepsilon$	1, 2	
30	$(\langle a^{d^*}\rangle\neg p \vee \langle a^{\times^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}}\rangle p)^\varepsilon$	3, 4	

31	$\langle a^{**}\rangle p^{\varepsilon}$	Any	No Proof
32	$\langle a^{\times^{\times}}\rangle p^{\varepsilon}$	Any	No Proof
33	$\langle a^{*^{*^{*^{*^{*^{*}}}}}}\rangle p^{\varepsilon}$	Any	No Proof
34	$\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p^{\varepsilon}$	Any	No Proof
35	$\langle a^{*^{\times^{*^{\times^{*^{\times}}}}}}\rangle p^{\varepsilon}$	Any	No Proof
36	$(\langle a^{d^{\times}} \sqcap b^{d^{\times}}\rangle \neg p \vee \langle (a \sqcup b)^{*}\rangle p)^{\varepsilon}$	1, 2	
36	$(\langle a^{d^{\times}} \sqcap b^{d^{\times}}\rangle \neg p \vee \langle (a \sqcup b)^{*}\rangle p)^{\varepsilon}$	3, 4	
37	$(\langle (a^{d} \sqcap b^{d})^{\times}\rangle \neg p \vee \langle (a^{*} ; b^{*})^{*}\rangle p)^{\varepsilon}$	1	

37	$(\langle(a^d \sqcap b^d)^\times\rangle\neg p \vee \langle(a^* ; b^*)^*\rangle p)^\varepsilon$	2	
37	$(\langle(a^d \sqcap b^d)^\times\rangle\neg p \vee \langle(a^* ; b^*)^*\rangle p)^\varepsilon$	3	
37	$(\langle(a^d \sqcap b^d)^\times\rangle\neg p \vee \langle(a^* ; b^*)^*\rangle p)^\varepsilon$	4	
38	$(\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle(a^* ; b)^\times\rangle p)^\varepsilon$	1, 2	

38	$(\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle(a^*\,;b)^\times\rangle p)^\varepsilon$	3, 4	 $[(\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle(a^*;b)^\times\rangle p)^{x_0}]^{x_0}$ mod_m $(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$ weak $(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}, (\langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$ weak $(\langle a^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}, (\langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$ weak $(\neg p)^\varepsilon, (\langle a^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}, (\langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$ $\vee$ $(\neg p)^\varepsilon, (\langle a^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b\rangle\langle(a^*;b)^\times\rangle p \vee \langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$ $*$ $(\neg p)^\varepsilon, (\langle a^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$ $\vee$ $(\neg p)^\varepsilon, (\langle a^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$ $\sqcup$ $(\neg p)^\varepsilon, (\langle(a^d \sqcup b^d)\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$ $\vee$ $(\neg p \vee \langle(a^d \sqcup b^d)\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$ $*$ $(\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$; $(\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle(a^*;b)\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$ $(\neg p)^\varepsilon, (p)^\varepsilon$ Ax1 $(\neg p)^\varepsilon, (p)^{x_0}$ exp $(\neg p)^\varepsilon, (\langle(a^d \sqcup b^d)\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (p)^{x_0}$ weak $(\neg p \vee \langle(a^d \sqcup b^d)\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (p)^{x_0}$ $\vee$ $(\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (p)^{x_0}$ $*$ $(\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (p \wedge \langle(a^*;b)\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$ $\wedge$ $(\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle(a^*;b)^\times\rangle p)^\varepsilon$ clo_{x_0} $(\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle(a^*;b)^\times\rangle p)^\varepsilon$ $\vee$
39	$(\langle(a^{d^*} \sqcup b^{d^*})^*\rangle\neg q \vee$ $\langle((p?\,;a^\times) \sqcup (\neg p?\,;b^\times))^\times\rangle q)^\varepsilon$	1	
39	$(\langle(a^{d^*} \sqcup b^{d^*})^*\rangle\neg q \vee$ $\langle((p?\,;a^\times) \sqcup (\neg p?\,;b^\times))^\times\rangle q)^\varepsilon$	2	
39	$(\langle(a^{d^*} \sqcup b^{d^*})^*\rangle\neg q \vee$ $\langle((p?\,;a^\times) \sqcup (\neg p?\,;b^\times))^\times\rangle q)^\varepsilon$	3	
39	$(\langle(a^{d^*} \sqcup b^{d^*})^*\rangle\neg q \vee$ $\langle((p?\,;a^\times) \sqcup (\neg p?\,;b^\times))^\times\rangle q)^\varepsilon$	4	

* Sequent 10 also has a map of closure sequents $(x_0 : \langle a\rangle p^\varepsilon)$, and a list of fixpoint sequents $[\langle a\rangle p^\varepsilon]$

B.2 More Detailed Proofs

This section covers some of the larger proofs from the table in more detail, in order to be able to make out the individual rule applications. We cover the proofs for sequent 30, 36, and half of the proofs for sequent 37. The remaining large proofs are so large that they would clutter this thesis more than offer any insights. That being said, the LaTeX code for these proofs can be found in the output folder in the GitHub repository.

First, let us show the proofs for sequent 30 ($((\langle a^{d^*}\rangle\neg p \vee \langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{\varepsilon})$.

What follows is the resulting proof for this sequent, using the Procrastination or Pre-Emptive mod_m Strategies. We will split up the proof, and the proof parts are linked together using subscripted $*$-symbols:

$$
\dfrac{\dfrac{\dfrac{\dfrac{[(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}]^{x_5}}{(\neg p \vee \langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}}\,\vee}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}}\,*}{(\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}}\,\mathrm{mod}_m}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}}\,\text{weak}
$$

$$*^{5}$$

$$
\dfrac{\dfrac{\dfrac{[(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4}]^{x_4}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}}\,\text{exp} \qquad *^{5}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p \wedge \langle a\rangle\langle a^{\times}\rangle\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}}\,\wedge}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4}}\,\mathrm{clo}_{x_5}
$$

$$*^{4}$$

$$
\dfrac{\dfrac{\dfrac{[(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3}]^{x_3}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4}}\,\text{exp} \qquad *^{4}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p \wedge \langle a^{\times}\rangle\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4}}\,\wedge}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3}}\,\mathrm{clo}_{x_4}
$$

$$*^{3}$$

$$
\dfrac{\dfrac{\dfrac{[(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2}]^{x_2}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3}}\,\text{exp} \qquad *^{3}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p \wedge \langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3}}\,\wedge}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2}}\,\mathrm{clo}_{x_3}
$$

$$*^{2}$$

$$\dfrac{\dfrac{\dfrac{[(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1}]^{x_1}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1,x_2}}\;\text{exp} \qquad *^{2}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p \wedge \langle a^{\times\times\times}\rangle\langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1,x_2}}\;\wedge}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1}}\;\text{clo}_{x_2}$$

$$*^{1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\;\text{Ax1}}{(\neg p)^{\varepsilon}, (p)^{x_0}}\;\text{exp}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (p)^{x_0}}\;\text{weak} \qquad \dfrac{\dfrac{\dfrac{[(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0}]^{x_0}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1}}\;\text{exp} \qquad *^{1}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times\times\times\times\times}\rangle p \wedge \langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1}}\;\wedge}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0}}\;\text{clo}_{x_1}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0}}\;\wedge}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times\times\times\times\times}\rangle p)^{\varepsilon}}\;\text{clo}_{x_0}}{(\neg p \vee \langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times\times\times\times\times}\rangle p)^{\varepsilon}}\;\vee}{\dfrac{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times\times\times\times\times}\rangle p)^{\varepsilon}}{(\langle a^{d^*}\rangle\neg p \vee \langle a^{\times\times\times\times\times\times}\rangle p)^{\varepsilon}}\;\vee}\;*$$

And let us also show the proof for the same sequent, using the Simple clo or Greedy Strategy:

$$\dfrac{\dfrac{\dfrac{\dfrac{[(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle\langle a^{\times\times}\rangle\langle a^{\times\times\times}\rangle\langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}]^{x_5}}{(\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle\langle a^{\times\times}\rangle\langle a^{\times\times\times}\rangle\langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}}\;\text{mod}_m}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle\langle a^{\times\times}\rangle\langle a^{\times\times\times}\rangle\langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}}\;\text{weak}}{(\neg p \vee \langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle\langle a^{\times\times}\rangle\langle a^{\times\times\times}\rangle\langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}}\;\vee}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a\rangle\langle a^{\times}\rangle\langle a^{\times\times}\rangle\langle a^{\times\times\times}\rangle\langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}}\;*$$

$$*^{5}$$

$$\dfrac{\dfrac{\dfrac{[(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times}\rangle\langle a^{\times\times\times}\rangle\langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1,x_2,x_3,x_4}]^{x_4}}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times}\rangle\langle a^{\times\times\times}\rangle\langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}}\;\text{exp} \qquad *^{5}}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times\times}\rangle\langle a^{\times\times\times}\rangle\langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p \wedge \langle a\rangle\langle a^{\times}\rangle\langle a^{\times\times}\rangle\langle a^{\times\times\times}\rangle\langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1,x_2,x_3,x_4,x_5}}\;\wedge}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times}\rangle\langle a^{\times\times}\rangle\langle a^{\times\times\times}\rangle\langle a^{\times\times\times\times}\rangle\langle a^{\times\times\times\times\times}\rangle\langle a^{\times\times\times\times\times\times}\rangle p)^{x_0,x_1,x_2,x_3,x_4}}\;\text{clo}_{x_5}$$

$$*^{4}$$

$$\cfrac{\cfrac{\cfrac{[(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3}]^{x_3}}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4}}\,\text{exp} \qquad *^4}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p \wedge \langle a^{\times}\rangle\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3,x_4}}\,\wedge}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3}}\,\text{clo}_{x_4}$$

$$*^3$$

$$\cfrac{\cfrac{\cfrac{[(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2}]^{x_2}}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3}}\,\text{exp} \qquad *^3}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p \wedge \langle a^{\times^{\times}}\rangle\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2,x_3}}\,\wedge}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2}}\,\text{clo}_{x_3}$$

$$*^2$$

$$\cfrac{\cfrac{\cfrac{[(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1}]^{x_1}}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2}}\,\text{exp} \qquad *^2}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p \wedge \langle a^{\times^{\times^{\times}}}\rangle\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1,x_2}}\,\wedge}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1}}\,\text{clo}_{x_2}$$

$$*^1$$

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\,\text{Ax1}}{(\neg p)^{\varepsilon}, (p)^{x_0}}\,\text{exp}}{(\neg p)^{\varepsilon}, (\langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (p)^{x_0}}\,\text{weak}}{(\neg p \vee \langle a^{d}\rangle\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (p)^{x_0}}\,\vee}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (p)^{x_0}}\,* \qquad \cfrac{\cfrac{\cfrac{[(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0}]^{x_0}}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1}}\,\text{exp} \qquad *^1}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p \wedge \langle a^{\times^{\times^{\times^{\times}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0,x_1}}\,\wedge}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0}}\,\text{clo}_{x_1}}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^{\times^{\times^{\times^{\times^{\times}}}}}\rangle\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{x_0}}\,\wedge}{(\langle a^{d^*}\rangle\neg p)^{\varepsilon}, (\langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{\varepsilon}}\,\text{clo}_{x_0}}{(\langle a^{d^*}\rangle\neg p \vee \langle a^{\times^{\times^{\times^{\times^{\times^{\times}}}}}}\rangle p)^{\varepsilon}}\,\vee$$

What follows is the proof for sequent 36 $((\langle a^{d^{\times}} \sqcap b^{d^{\times}}\rangle\neg p \vee \langle (a \sqcup b)^*\rangle p)^{\varepsilon})$, obtained with the Procrastination or Pre-Emptive mod_m Strategy, split into 3 parts:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^\varepsilon, (p)^\varepsilon}\,\text{Ax1}}{(\neg p)^\varepsilon, (p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{weak}}{(\neg p)^\varepsilon, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{weak}}{(\neg p)^{x_0}, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{exp} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[(\langle a^{d^\times}\rangle\neg p)^{x_0}, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon]^{x_0}}{(\langle a^{d^\times}\rangle\neg p)^{x_0}, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p \vee \langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\vee}{(\langle a^{d^\times}\rangle\neg p)^{x_0}, (p)^\varepsilon, (\langle(a\sqcup b)\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\sqcup}{(\langle a^{d^\times}\rangle\neg p)^{x_0}, (p \vee \langle(a\sqcup b)\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\vee}{(\langle a^{d^\times}\rangle\neg p)^{x_0}, (\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,*}{(\langle a^d\rangle\langle a^{d^\times}\rangle\neg p)^{x_0}, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{mod}_m}{(\langle a^d\rangle\langle a^{d^\times}\rangle\neg p)^{x_0}, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{weak}}{(\langle a^d\rangle\langle a^{d^\times}\rangle\neg p)^{x_0}, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{weak}}{(\neg p \wedge \langle a^d\rangle\langle a^{d^\times}\rangle\neg p)^{x_0}, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\wedge$$

$$*^1$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^\varepsilon, (p)^\varepsilon}\,\text{Ax1}}{(\neg p)^\varepsilon, (p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{weak}}{(\neg p)^\varepsilon, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{weak}}{(\neg p)^{x_0}, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{exp} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[(\langle b^{d^\times}\rangle\neg p)^{x_0}, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon]^{x_0}}{(\langle b^{d^\times}\rangle\neg p)^{x_0}, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p \vee \langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\vee}{(\langle b^{d^\times}\rangle\neg p)^{x_0}, (p)^\varepsilon, (\langle(a\sqcup b)\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\sqcup}{(\langle b^{d^\times}\rangle\neg p)^{x_0}, (p \vee \langle(a\sqcup b)\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\vee}{(\langle b^{d^\times}\rangle\neg p)^{x_0}, (\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,*}{(\langle b^d\rangle\langle b^{d^\times}\rangle\neg p)^{x_0}, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{mod}_m}{(\langle b^d\rangle\langle b^{d^\times}\rangle\neg p)^{x_0}, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{weak}}{(\langle b^d\rangle\langle b^{d^\times}\rangle\neg p)^{x_0}, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{weak}}{(\neg p \wedge \langle b^d\rangle\langle b^{d^\times}\rangle\neg p)^{x_0}, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\wedge$$

$$*^2$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*^1}{(\langle a^{d^\times}\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{clo}_{x_0}}{(\langle a^{d^\times}\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p \vee \langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\vee}{(\langle a^{d^\times}\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle(a\sqcup b)\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\sqcup}{(\langle a^{d^\times}\rangle\neg p)^\varepsilon, (p \vee \langle(a\sqcup b)\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\vee}{(\langle a^{d^\times}\rangle\neg p)^\varepsilon, (\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,* \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{*^2}{(\langle b^{d^\times}\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon, (\langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\text{clo}_{x_0}}{(\langle b^{d^\times}\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle a\rangle\langle(a\sqcup b)^*\rangle p \vee \langle b\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\vee}{(\langle b^{d^\times}\rangle\neg p)^\varepsilon, (p)^\varepsilon, (\langle(a\sqcup b)\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\sqcup}{(\langle b^{d^\times}\rangle\neg p)^\varepsilon, (p \vee \langle(a\sqcup b)\rangle\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\vee}{(\langle b^{d^\times}\rangle\neg p)^\varepsilon, (\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,*}{(\langle a^{d^\times}\rangle\neg p \wedge \langle b^{d^\times}\rangle\neg p)^\varepsilon, (\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\wedge}{(\langle(a^{d^\times} \sqcap b^{d^\times})\rangle\neg p)^\varepsilon, (\langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\sqcap}{(\langle(a^{d^\times} \sqcap b^{d^\times})\rangle\neg p \vee \langle(a\sqcup b)^*\rangle p)^\varepsilon}\,\vee$$

And here is the proof for the same sequent, but obtained with the Simple clo or Greedy Strategy, again split into 3 parts:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\,\text{Ax1}}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle (a \sqcup b)\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\text{weak}}{(\neg p)^{x_0}, (p)^{\varepsilon}, (\langle (a \sqcup b)\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\text{exp}}{(\neg p)^{x_0}, (p \vee \langle (a \sqcup b)\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\vee}{(\neg p)^{x_0}, (\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,* \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[(\langle a^{d^{\times}}\rangle \neg p)^{x_0}, (\langle (a \sqcup b)^*\rangle p)^{\varepsilon}]^{x_0}}{(\langle a^d\rangle\langle a^{d^{\times}}\rangle \neg p)^{x_0}, (\langle a\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\text{mod}_m}{(\langle a^d\rangle\langle a^{d^{\times}}\rangle \neg p)^{x_0}, (\langle a\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\text{weak}}{(\langle a^d\rangle\langle a^{d^{\times}}\rangle \neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\text{weak}}{(\langle a^d\rangle\langle a^{d^{\times}}\rangle \neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle (a \sqcup b)^*\rangle p \vee \langle b\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\vee}{(\langle a^d\rangle\langle a^{d^{\times}}\rangle \neg p)^{x_0}, (p)^{\varepsilon}, (\langle (a \sqcup b)\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\sqcup}{(\langle a^d\rangle\langle a^{d^{\times}}\rangle \neg p)^{x_0}, (p \vee \langle (a \sqcup b)\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\vee}{(\langle a^d\rangle\langle a^{d^{\times}}\rangle \neg p)^{x_0}, (\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,*}{(\neg p \wedge \langle a^d\rangle\langle a^{d^{\times}}\rangle \neg p)^{x_0}, (\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\wedge$$

$$*^1$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\,\text{Ax1}}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle (a \sqcup b)\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\text{weak}}{(\neg p)^{x_0}, (p)^{\varepsilon}, (\langle (a \sqcup b)\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\text{exp}}{(\neg p)^{x_0}, (p \vee \langle (a \sqcup b)\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\vee}{(\neg p)^{x_0}, (\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,* \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[(\langle b^{d^{\times}}\rangle \neg p)^{x_0}, (\langle (a \sqcup b)^*\rangle p)^{\varepsilon}]^{x_0}}{(\langle b^d\rangle\langle b^{d^{\times}}\rangle \neg p)^{x_0}, (\langle b\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\text{mod}_m}{(\langle b^d\rangle\langle b^{d^{\times}}\rangle \neg p)^{x_0}, (\langle a\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\text{weak}}{(\langle b^d\rangle\langle b^{d^{\times}}\rangle \neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\text{weak}}{(\langle b^d\rangle\langle b^{d^{\times}}\rangle \neg p)^{x_0}, (p)^{\varepsilon}, (\langle a\rangle\langle (a \sqcup b)^*\rangle p \vee \langle b\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\vee}{(\langle b^d\rangle\langle b^{d^{\times}}\rangle \neg p)^{x_0}, (p)^{\varepsilon}, (\langle (a \sqcup b)\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\sqcup}{(\langle b^d\rangle\langle b^{d^{\times}}\rangle \neg p)^{x_0}, (p \vee \langle (a \sqcup b)\rangle\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\vee}{(\langle b^d\rangle\langle b^{d^{\times}}\rangle \neg p)^{x_0}, (\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,*}{(\neg p \wedge \langle b^d\rangle\langle b^{d^{\times}}\rangle \neg p)^{x_0}, (\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\wedge$$

$$*^2$$

$$\dfrac{\dfrac{\dfrac{\dfrac{*^1}{(\langle a^{d^{\times}}\rangle \neg p)^{\varepsilon}, (\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\text{clo}_{x_0} \qquad \dfrac{*^2}{(\langle b^{d^{\times}}\rangle \neg p)^{\varepsilon}, (\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\text{clo}_{x_0}}{(\langle a^{d^{\times}}\rangle \neg p \wedge \langle b^{d^{\times}}\rangle \neg p)^{\varepsilon}, (\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\wedge}{(\langle (a^{d^{\times}} \sqcap b^{d^{\times}})\rangle \neg p)^{\varepsilon}, (\langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\sqcap}{(\langle (a^{d^{\times}} \sqcap b^{d^{\times}})\rangle \neg p \vee \langle (a \sqcup b)^*\rangle p)^{\varepsilon}}\,\vee$$

What follows is the proof for sequent 37 ($(\langle (a^d \sqcap b^d)^{\times}\rangle \neg p \vee \langle (a^* ; b^*)^*\rangle p)^{\varepsilon}$) for the Procrastination Strategy:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}}\,\text{Ax1}}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle (a^*; b^*)^*\rangle p)^{\varepsilon}}\,\text{weak}}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle (a^*; b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle (a^*; b^*)^*\rangle p)^{\varepsilon}}\,\text{weak}}{(\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle (a^*; b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle (a^*; b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle (a^*; b^*)^*\rangle p)^{\varepsilon}}\,\text{weak}}{(\neg p)^{x_0}, (p)^{\varepsilon}, (\langle (a^*; b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle (a^*; b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle (a^*; b^*)^*\rangle p)^{\varepsilon}}\,\text{exp}$$

$$*^1$$

$$\begin{array}{cl}
[(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle(a^*;b^*)^*\rangle p\vee\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}]^{x_0} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle b^*\rangle\langle(a^*;b^*)^*\rangle p\vee\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & ; \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p\vee\langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle(a^*;b^*)^*\rangle p\vee\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle b^*\rangle\langle(a^*;b^*)^*\rangle p\vee\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \mathrm{mod}_m \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \text{weak} \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \text{weak} \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle(a^*;b^*)^*\rangle p\vee\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \text{weak} \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle(a^*;b^*)^*\rangle p\vee\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle b^*\rangle\langle(a^*;b^*)^*\rangle p\vee\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & ; \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(p\vee\langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\
*^2 &
\end{array}$$

$$\begin{array}{cl}
[(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle(a^*;b^*)^*\rangle p\vee\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}]^{x_0} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle b^*\rangle\langle(a^*;b^*)^*\rangle p\vee\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & ; \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p\vee\langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle(a^*;b^*)^*\rangle p\vee\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \mathrm{mod}_m \\ \hline
(\langle b^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \text{weak} \\ \hline
(\langle b^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \text{weak} \\ \hline
(\langle b^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(\langle(a^*;b^*)^*\rangle p\vee\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \text{weak} \\ \hline
(\langle b^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle(a^*;b^*)^*\rangle p\vee\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle b^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle b^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle b^*\rangle\langle(a^*;b^*)^*\rangle p\vee\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle b^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & ; \\ \hline
(\langle b^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle b^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(p\vee\langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle b^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0},(p)^{\varepsilon},(\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon},(\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\
*^3 &
\end{array}$$

$$\begin{array}{cl}
*^2 \qquad *^3 & \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p \wedge \langle b^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \wedge \\ \hline
*^1 \qquad (\langle(a^d\sqcap b^d)\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \sqcap \\ \hline
(\neg p \wedge \langle(a^d\sqcap b^d)\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \wedge \\ \hline
(\neg p \wedge \langle(a^d\sqcap b^d)\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle(a^*;b^*)^*\rangle p \vee \langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle(a^*;b^*)^*\rangle p \vee \langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \mathrm{clo}_{x_0} \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle b^*\rangle\langle(a^*;b^*)^*\rangle p \vee \langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & ; \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (p \vee \langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle(a^*;b^*)^*\rangle p \vee \langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle b^*\rangle\langle(a^*;b^*)^*\rangle p \vee \langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p)^{\varepsilon}, (\langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & ; \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (p \vee \langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{\varepsilon}, (\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & *
\end{array}$$

For the same sequent, we can find a proof by using the Pre-Emptive mod_m Strategy. The bottom of the proof stays the same, only the two upper branches shrink in size. These two new upper branches are shown below:

$$\begin{array}{cl}
[(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle(a^*;b^*)^*\rangle p \vee \langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}]^{x_0} & \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle b^*\rangle\langle(a^*;b^*)^*\rangle p \vee \langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & ; \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p \vee \langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle(a^*;b^*)^*\rangle p \vee \langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle b^*\rangle\langle(a^*;b^*)^*\rangle p \vee \langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \vee \\ \hline
(\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & * \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \mathrm{mod}_m \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \text{weak} \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \text{weak} \\ \hline
(\langle a^d\rangle\langle(a^d\sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \text{weak} \\
*^2 &
\end{array}$$

$$\begin{array}{cl}
[(\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle(a^*;b^*)^*\rangle p \vee \langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}]^{x_0} & \\ \cline{1-1}
 & * \\
(\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\ \cline{1-1}
 & \vee \\
(\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle b^*\rangle\langle(a^*;b^*)^*\rangle p \vee \langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\ \cline{1-1}
 & * \\
(\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\ \cline{1-1}
 & ; \\
(\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\ \cline{1-1}
 & \vee \\
(\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p \vee \langle(a^*;b^*)\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\ \cline{1-1}
 & * \\
(\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\ \cline{1-1}
 & \vee \\
(\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle(a^*;b^*)^*\rangle p \vee \langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\ \cline{1-1}
 & * \\
(\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\ \cline{1-1}
 & \mathrm{mod}_m \\
(\langle b^d\rangle\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\ \cline{1-1}
 & \text{weak} \\
(\langle b^d\rangle\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\ \cline{1-1}
 & \text{weak} \\
(\langle b^d\rangle\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\ \cline{1-1}
 & \text{weak} \\
(\langle b^d\rangle\langle(a^d \sqcap b^d)^{\times}\rangle\neg p)^{x_0}, (p)^{\varepsilon}, (\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle b\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon}, (\langle a\rangle\langle a^*\rangle\langle b^*\rangle\langle(a^*;b^*)^*\rangle p)^{\varepsilon} & \\
*^3 &
\end{array}$$

C Rascal Code

C.1 CloG_Base

C.1.1 CloGSyntax.rsc

```
module CloG_Base::CloGSyntax
/*
 * A module defining the concrete syntax for the input of a CloG sequent
 *
 * Converted to abstract syntax tree by CST2AST_CloG module
 */

extend lang::std::Layout;

/*
 * A seq file starts with "Seq" and consists of a list of labeled formulae, represented as
     ↪  CloG terms wrapped by [].
 */
start syntax SCloGSequent
       = "Seq" "[" SCloGTerm* seq "]";

/*
 * Each CloG term is defined as a CloG expression (a game logic formula in normal form)
     ↪ with a superscript label
 *
 * The label is defined as a list of named Id's wrapped by [].
 */
syntax SCloGTerm
       = SCloGExpr ex "^" "[" SId* label "]";

// Syntax definition of normal form game logic expression used in CloG
syntax SCloGExpr
       = prop: SId p
       | not: "~" SId p
       > left par: "(" SCloGExpr ex ")"
       > right strat: "\<" SCloGGame g "\>" SCloGExpr ex
       > left and: SCloGExpr exL "&" SCloGExpr exR
       > left or: SCloGExpr exL "|" SCloGExpr exR;

// Syntax definition of normal form game expression used in CloG
syntax SCloGGame
       = agame: SId g
       | dual: SId g "^d"
       > left par: "(" SCloGGame ga ")"
       > left \test: SCloGExpr ga "?"
       > left dTest: SCloGExpr ga "!"
       > left iter: SCloGGame ga "*"
       > left dIter: SCloGGame ga "^x"
       > left con: SCloGGame gaL ";" SCloGGame gaR
       > left dChoice: SCloGGame gaL "&&" SCloGGame gaR
       > left choice: SCloGGame gaL "||" SCloGGame gaR;

// Syntax definition of a named ID which can have a subscript integer
// Used for atomic games, atomic propositions, and Clo rule names
syntax SId
       = Id n "_" Int sub
       | Id n;

// Regular Expression definition for an integer that can be used for an ID subscript
lexical Int
       = [1-9][0-9]*
       | [0];

// To avoid ambiguity with the underscore, we define an identifier as only consisting
// of letters (which is already more liberal than the single letter restraint in the
// literature.
lexical Id = [a-z A-Z]+
    ;
```

C.1.2 CST2AST_CloG.rsc

```
module CloG_Base::CST2AST_CloG
/*
 * A module containing functions to transform the CloG input to an abstract syntax tree
 */

import String;

import CloG_Base::CloGSyntax;
import CloG_Base::GLASTs;

/* Main function for syntax conversion
 *
 * Input: Parsed CloG input sequent syntax
 * Output: CloGSequent algebraic type for whole proof
 */
CloGSequent cst2astCloG(start[SCloGSequent] ss){
       SCloGSequent s = ss.top;

       return [ cst2astCloG(t) | SCloGTerm t <- s.seq ];
}

/* Function for syntax conversion of a CloG term. Conversion called on the proposition
    ↪ and list reduction used to convert label
 *
 * Input: Concrete syntax CloG term
 * Output: CloG term algebraic type
 */
CloGTerm cst2astCloG(SCloGTerm t){
       return term(cst2astCloG(t.ex), [id2name(n) | SId n <- t.label], false);
}

/* Function for syntax conversion of a normal form game logic propositional formula.
    ↪ Switch statement used to identify which
 * operator is used.
 *
 * Input: Concrete syntax game expression
 * Output: Game logic proposition algebraic type
 */
GameLog cst2astCloG(SCloGExpr exp){
        switch (exp){
                case(SCloGExpr)`~ <SId p>`:
                        return neg(atomP(id2prop(p)));
                case(SCloGExpr)`<SId p>`:
                        return atomP(id2prop(p));
                case(SCloGExpr)`( <SCloGExpr ex> )`:
                        return cst2astCloG(ex);
                case(SCloGExpr)`\< <SCloGGame g> \> <SCloGExpr ex>`:
                        return \mod(cst2astCloG(g), cst2astCloG(ex));
                case(SCloGExpr)`<SCloGExpr exL> & <SCloGExpr exR>`:
                        return and(cst2astCloG(exL), cst2astCloG(exR));
                case(SCloGExpr)`<SCloGExpr exL> | <SCloGExpr exR>`:
                        return or(cst2astCloG(exL), cst2astCloG(exR));
                default: throw "Unsupported Game Logic Formula";
        }
}

/* Function for syntax conversion of a normal form game formula. Switch statement used to
  ↪  identify which
 * operator is used.
 *
 * Input: Concrete syntax game expression
 * Output: Game algebraic type
 */
Game cst2astCloG(SCloGGame g){
        switch (g){
                case (SCloGGame)`<SId a> ^d`:
                        return dual(atomG(id2agame(a)));
                case (SCloGGame)`<SId a>`:
                        return atomG(id2agame(a));
                case (SCloGGame)`( <SCloGGame ga> )`:
                        return cst2astCloG(ga);
                case (SCloGGame)`<SCloGExpr ex> ?`:
                        return \test(cst2astCloG(ex));
                case (SCloGGame)`<SCloGExpr ex> !`:
                        return dTest(cst2astCloG(ex));
                case (SCloGGame)`<SCloGGame ga> *`:
                        return iter(cst2astCloG(ga));
                case (SCloGGame)`<SCloGGame ga> ^x`:
                        return dIter(cst2astCloG(ga));
                case (SCloGGame)`<SCloGGame gaL> ; <SCloGGame gaR>`:
                        return concat(cst2astCloG(gaL), cst2astCloG(gaR));
                case (SCloGGame)`<SCloGGame gaL> && <SCloGGame gaR>`:
                        return dChoice(cst2astCloG(gaL), cst2astCloG(gaR));
                case (SCloGGame)`<SCloGGame gaL> || <SCloGGame gaR>`:
                        return choice(cst2astCloG(gaL), cst2astCloG(gaR));
                default: throw "Unsupported Game Formula";
        }
}
```

```
/* ID conversion functions need different names as they use the same concrete syntax type
    ↪ . Is more modular
 * than a larger switch statement.
 *
 * Input: Concrete syntax for an identifier
 * Output: Algebraic the corresponding atom or name
 */

// Function for syntax conversion of a Clo rule name
CloGName id2name(SId n){
        switch (n) {
                case (SId)`<Id id> _ <Int sub>`:
                        return nameS("<id>",toInt("<sub>"));
                case (SId)`<Id id>`:
                        return name("<id>");
                default: throw "Unsupported Name";
        }
}

// Function for syntax conversion of an atomic proposition
Prop id2prop(SId p){
        switch (p){
                case (SId)`<Id id> _ <Int sub>`:
                        return propS("<id>",toInt("<sub>"));
                case (SId)`<Id id>`:
                        return prop("<id>");
                default: throw "Unsupported Proposition";
        }
}

// Function for syntax conversion of an atomic game
AGame id2agame(SId a){
        switch (a){
                case (SId)`<Id id> _ <Int sub>`:
                        return agameS("<id>",toInt("<sub>"));
                case (SId)`<Id id>`:
                        return agame("<id>");
                default: throw "Unsupported Proposition";
        }
}
```

C.1.3 GLASTs.rsc

```
module CloG_Base::GLASTs
/*
 * A module defining Abstract Syntax Types for CloG sequents and proofs
 */

/*
 * Abstract Syntax for a CloG Proof and CloG Sequent
 */
```

```
data CloGProof(loc src = |tmp:///|)
       = CloGLeaf()
       | disClo(list[CloGTerm] seq, CloGName n)
       | CloGUnaryInf(list[CloGTerm] seq, CloGRule rule, CloGProof inf)
       | CloGBinaryInf(list[CloGTerm] seq, CloGProof infL, CloGProof infR);

alias CloGSequent = list[CloGTerm];

data CloGTerm(loc src = |tmp:///|)
       = term(GameLog s, list[CloGName] label, bool active);

data CloGRule(loc src = |tmp:///|)
       = ax1()
       | modm()
       | andR()
       | orR()
       | choiceR()
       | dChoiceR()
       | weak()
       | exp()
       | concatR()
       | iterR()
       | testR()
       | dIterR()
       | dTestR()
       | clo(CloGName n);

data CloGName(loc src = |tmp:///|)
       = name(str l)
       | nameS(str l, int sub);

/*
 * Abstract Syntax for a Game Logic Formula
 */

data GameLog(loc src = |tmp:///|)
       = top()
       | bot()
       | atomP(Prop p)
       | neg(GameLog pr)
       | \mod(Game g, GameLog pr)
       | modExp(Game g, GameLog pr, int n) // fp annotation for expansion rule
       | dMod(Game g, GameLog pr)
       | and(GameLog pL, GameLog pR)
       | or(GameLog pL, GameLog pR)
       | cond(GameLog pL, GameLog pR)
       | biCond(GameLog pL, GameLog pR);

data Game(loc src = |tmp:///|)
       = atomG(AGame g)
       | dual(Game ga)
       | \test(GameLog p)
       | dTest(GameLog p)
       | iter(Game ga)
       | dIter(Game ga)
       | dIterExp(Game ga, int n) // fp annotation for expansion rule
       | concat(Game gL, Game gR)
       | choice(Game gL, Game gR)
       | dChoice(Game gL, Game gR);

/*
 * Abstract syntax for atomic games and propositions
 */

data AGame(loc src = |tmp:///|)
       = agame(str l)
       | agameS(str l, int sub);

data Prop(loc src = |tmp:///|)
       = prop(str l)
       | propS(str l, int sub);
```

C.1.4 LaTeXOutput.rsc

```
module CloG_Base::LaTeXOutput
/*
 * A module defining the transformation from abstract proof trees to LaTeX proof trees
 */

import CloG_Base::GLASTs;

import IO;
import List;

// Functions takes a proof tree and file location as input then writes the LaTeX output
    ↪ to the file.
void LaTeXOutput(CloGProof p, loc out){
       writeFile(out, LaTeXOutput(p));
}

// Functions define the preamle and proof tree wrapper for LaTeX output
str LaTeXOutput(CloGProof p) =
       "\\documentclass{article}
       ’\\usepackage[utf8]{inputenc}
       ’\\usepackage{prooftrees}
       ’
       ’\\begin{document}
       ’
       ’\\begin{prooftree}
       ’<LaTeXCloGTree(p)>
       ’\\end{prooftree}
       ’
       ’\\end{document}";

```

```
// Function to output the LaTeX proof tree part for each CloG sequent and assocated rule
    ↪ label
str LaTeXCloGTree(CloGLeaf()) =
       "\\AxiomC{}";
str LaTeXCloGTree(disClo(list[CloGTerm] seq, CloGName n)) =
       "\\AxiomC{$[<(LaTeXCloGTree(head(seq)) | it + ", " + LaTeXCloGTree(t) | CloGTerm t
           ↪  <- tail(seq))>]^{<LaTeXCloGTree(n)>}$}";
str LaTeXCloGTree(CloGUnaryInf(list[CloGTerm] seq, CloGRule rule, CloGProof inf)) =
       "<LaTeXCloGTree(inf)>
       '\\RightLabel{<LaTeXCloGTree(rule)>}
       '\\UnaryInfC{$<(LaTeXCloGTree(head(seq)) | it + ", " + LaTeXCloGTree(t) | CloGTerm
           ↪  t <- tail(seq))>$}";
str LaTeXCloGTree(CloGBinaryInf(list[CloGTerm] seq, CloGProof infL, CloGProof infR)) =
       "<LaTeXCloGTree(infL)>
       '<LaTeXCloGTree(infR)>
       '\\RightLabel{$\\wedge$}
       '\\BinaryInfC{$<(LaTeXCloGTree(head(seq)) | it + ", " + LaTeXCloGTree(t) |
           ↪ CloGTerm t <- tail(seq))>$}";

// Function to output the superscript label attached to each logic formula and to
// specify the color of the term depending on whether it is active or not
str LaTeXCloGTree(term(GameLog s, [], bool active))
       = "<active ? "\\textcolor{red}{" : "">(<LaTeXGameLog(s)>)^{\\varepsilon}<active ?
           ↪ "}" : "">";
str LaTeXCloGTree(term(GameLog s, list[CloGName] label, bool active))
       = "<active ? "\\textcolor{red}{" : "">(<LaTeXGameLog(s)>)^{<(LaTeXCloGTree(head(
           ↪ label)) | it
         + ", " + LaTeXCloGTree(n) | CloGName n <- tail(label))>} <active ? "}" : "">";

// Function to output the CloG rule labels
str LaTeXCloGTree(ax1()) = "Ax1";
str LaTeXCloGTree(modm()) = "mod$_{m}$";
str LaTeXCloGTree(andR()) = "$\\wedge$";
str LaTeXCloGTree(orR()) = "$\\vee$";
str LaTeXCloGTree(choiceR()) = "$\\sqcup$";
str LaTeXCloGTree(dChoiceR()) = "$\\sqcap$";
str LaTeXCloGTree(weak()) = "weak";
str LaTeXCloGTree(exp()) = "exp";
str LaTeXCloGTree(concatR()) = "$;$";
str LaTeXCloGTree(iterR()) = "$*$";
str LaTeXCloGTree(testR()) = "$?$";
str LaTeXCloGTree(dIterR()) = "$\\times$";
str LaTeXCloGTree(dTestR()) = "$!$";
str LaTeXCloGTree(clo(CloGName n)) = "clo$_{<LaTeXCloGTree(n)>}$";

// Function to output a CloG name which can have a subscript
str LaTeXCloGTree(name(str n)) = "<n>";
str LaTeXCloGTree(nameS(str n, int sub)) = "<n>_{<sub>}";

// Function to output the game logic formulae in LaTeX maths mode
str LaTeXGameLog(top()) = "(p\\vee\\neg p)";
str LaTeXGameLog(bot()) = "(p\\wedge\\neg p)";
str LaTeXGameLog(atomP(Prop p)) = "<LaTeXGameLog(p)>";
str LaTeXGameLog(neg(GameLog pr)) = "\\neg <hasPar(pr)>";
str LaTeXGameLog(\mod(Game g, GameLog pr)) = "\\langle <hasPar(g)>\\rangle <hasPar(pr)>";
str LaTeXGameLog(dMod(Game g, GameLog pr)) = "[<hasPar(g)>]<hasPar(pr)>";
str LaTeXGameLog(and(GameLog pL, GameLog pR)) = "<hasPar(pL)>\\wedge <hasPar(pR)>";
str LaTeXGameLog(or(GameLog pL, GameLog pR)) = "<hasPar(pL)>\\vee <hasPar(pR)>";
str LaTeXGameLog(cond(GameLog pL, GameLog pR)) = "<hasPar(pL)>\\rightarrow <hasPar(pR)>";
str LaTeXGameLog(biCond(GameLog pL, GameLog pR)) = "<hasPar(pL)>\\leftrightarrow <hasPar(
    ↪ pR)>";

// Function to output the game formulae in LaTeX maths mode
str LaTeXGameLog(atomG(AGame g)) = "<LaTeXGameLog(g)>";
str LaTeXGameLog(dual(Game ga)) = "{<hasPar(ga)>}^d";
str LaTeXGameLog(\test(GameLog g)) = "<hasPar(g)>?";
str LaTeXGameLog(dTest(GameLog g)) = "<hasPar(g)>!";
str LaTeXGameLog(iter(Game ga)) = "{<hasPar(ga)>}^{*}";
str LaTeXGameLog(dIter(Game ga)) = "{<hasPar(ga)>}^{\\times}";
str LaTeXGameLog(concat(Game gL, Game gR)) = "<hasPar(gL)>;<hasPar(gR)>";
str LaTeXGameLog(choice(Game gL, Game gR)) = "<hasPar(gL)>\\sqcup <hasPar(gR)>";
str LaTeXGameLog(dChoice(Game gL, Game gR)) = "<hasPar(gL)>\\sqcap <hasPar(gR)>";

// Function to put parentheses around only the binary connectives and not unary
    ↪ connectives or atomic formulae
str hasPar(top()) = "(p\\vee\\neg p)";
str hasPar(bot()) = "(p\\wedge\\neg p)";
str hasPar(atomP(Prop p)) = "<LaTeXGameLog(atomP(p))>";
str hasPar(neg(GameLog pr)) = "<LaTeXGameLog(neg(pr))>";
str hasPar(\mod(Game g, GameLog pr)) = "<LaTeXGameLog(\mod(g,pr))>";
str hasPar(dMod(Game g, GameLog pr)) = "<LaTeXGameLog(dMod(g,pr))>";
str hasPar(and(GameLog pL, GameLog pR)) = "(<LaTeXGameLog(and(pL,pR))>)";
str hasPar(or(GameLog pL, GameLog pR)) = "(<LaTeXGameLog(or(pL,pR))>)";
str hasPar(cond(GameLog pL, GameLog pR)) = "(<LaTeXGameLog(cond(pL,pR))>)";
str hasPar(biCond(GameLog pL, GameLog pR)) = "(<LaTeXGameLog(biCond(pL,pR))>)";
str hasPar(atomG(AGame g)) = "<LaTeXGameLog(atomG(g))>";
str hasPar(dual(Game ga)) = "<LaTeXGameLog(dual(ga))>";
str hasPar(\test(GameLog g)) = "<LaTeXGameLog(\test(g))>";
str hasPar(dTest(GameLog g)) = "<LaTeXGameLog(dTest(g))>";
str hasPar(iter(Game ga)) = "<LaTeXGameLog(iter(ga))>";
str hasPar(dIter(Game ga)) = "<LaTeXGameLog(dIter(ga))>";
str hasPar(concat(Game gL, Game gR)) = "(<LaTeXGameLog(concat(gL,gR))>)";
str hasPar(choice(Game gL, Game gR)) = "(<LaTeXGameLog(choice(gL,gR))>)";
str hasPar(dChoice(Game gL, Game gR)) = "(<LaTeXGameLog(dChoice(gL,gR))>)";

// Functions to output the atomic propositions and games which can have a subscript
str LaTeXGameLog(agame(str n)) = "<n>";
str LaTeXGameLog(agameS(str n, int sub)) = "<n>_{<sub>}";
str LaTeXGameLog(prop(str n)) = "<n>";
str LaTeXGameLog(propS(str n, int sub)) = "<n>_{<sub>}";
```

C.2 ATP

C.2.1 ATP_Base.rsc

```
module ATP::ATP_Base
```

```
/*
 * A module defining data types relevant for the Automated Theorem Prover, and
 * some helper functions.
 */

import CloG_Base::GLASTs;
import Exception;

/*
 * A FpSeq, or fixpoint sequent is a tuple containing a sequent that contains
 * a specific fixpoint formula, and the index of the term in that sequent which
 * contains the fixpoint formula.
 */
alias FpSeq = tuple[CloGSequent contextSeq, int fpFormulaIdx];

/*
 * A CloSeqs is a map, mapping each name associated with a fixpoint formula to the
 * FpSeq containing that formula at the specified index.
 */
alias CloSeqs = map[CloGName name, FpSeq fpSeq];

/*
 * MaybeProof is either a CloGProof, noProof(), which indicates no proof could be
 * found, or cantApply(), which indicates a rule could not be applied
 */
data MaybeProof
       = proof(CloGProof p)
       | noProof()
       | cantApply();

/*
 * MaybeSequent is either a CloGSequent, or noSeq(), which indicates no sequent could
 * be derived.
 */
data MaybeSequent
       = sequent(CloGSequent seq)
       | noSeq();

/*
 * MaybeSequents is either a pair of CloGSequents, or noSeqs(), which indicates no
 * sequents could be derived.
 */
data MaybeSequents
   = sequents(CloGSequent left, CloGSequent right)
   | noSeqs();

/*
 * An algorithm returning whether one fixpoint formula is less than or equal to
 * the other, according to the fixpoint ordering defined in the literature.
 *
 * Input: two GameLog formulae
 * Output: a bool, true if the left GameLog formula is less than or equal to the
 * right GameLog formula according to the fixpoint ordering, and false
 * otherwise.
 *
 * One fixpoint formula is considered less than or equal to another fixpoint
 * formula, if the game in the other fixpoint formula is a subterm of the game
 * in the one fixpoint formula.
 */
bool fpLessThanOrEqualTo(GameLog left, GameLog right) {
       if (\mod(Game fp0, _) := left && \mod(Game fp1, _) := right)
              if (
              (iter(_) := fp0 || dIter(_) := fp0)
        && (iter(_) := fp1 || dIter(_) := fp1)
              )
                     return subTerm(fp1, fp0);
       throw IllegalArgument("Error: cannot apply fixpoint ordering on non-fixpoint
           ↪ formulae!");
}

/*
 * An algorithm returning whether one game formula is a subterm of the other (or
 * if they are equal).
 *
 * Input: two Game formulae
 * Output: a bool, true if the left Game formula is a subterm of the right, and
 * false otherwise.
 *
 * One Game formula is a subterm of the other, if the one formula appears in the
 * other formula, which is the same as saying the one Game formula is a descendant
 * of the other.
 */
bool subTerm(Game g, Game h) {
       if (/g := h)
              return true;
       return false;
}
```

C.2.2 ProofSearch.rsc

```
module ATP::ProofSearch
/*
 * Module defining the basic proof search algorithm, and some helper functions
 */

import List;
import Map;
import CloG_Base::GLASTs;
import ATP::ATP_Base;
import ATP::RuleApplications;

/*
 * Calls the main proof search algorithm without a maximum recursion depth, and with an
 * initially empty map of closure sequents and list of fixpoint sequents.
 */
MaybeProof proofSearch(CloGSequent seq) {
       return proofSearch(seq, (), [], -1);
}

/*
 * Calls the main proof search algorithm without a maximum recursion depth.
 */
MaybeProof proofSearch(CloGSequent seq, CloSeqs cloSeqs, list[CloGSequent] fpSeqs) {
       return proofSearch(seq, cloSeqs, fpSeqs, -1);
}

/*
 * Calls the main proof search algorithm with an initially empty map of closure sequents
     ↪ and
 * list of fixpoint sequents.
 */
MaybeProof proofSearch(CloGSequent seq, int maxRecursionDepth) {
       return proofSearch(seq, (), [], maxRecursionDepth);
}

/*
 * A depth-first proof search algorithm, applying a saturation strategy.
 *
 * Input: a sequent to be proven, a map which maps the names present in the sequent to the
 * associated fixpoint formulae, a list of sequents associated with earlier applications
 * of iter, dIter, and clo rules, and an integer representing how deep in the proof tree
 * we are.
 * Output: a MaybeProof, which is either a CloGProof, a noProof() which essentially acts
     ↪ as
 * a null value, indicating no proof could be found, or a cantApply() which is mostly
 * just used to keep track of where to backtrack to.
 *
 * When the depth reaches 0, we do not recurse any further, and noProof() is returned.
 *
 * Since terms in the sequent are saved in a list, rather than the theoretical set, we
     ↪ then
 * remove the duplicates from this list.
 *
 * First, we try discharging a closure sequent if possible.
 *
 * If this is not possible, we try to detect cycles. If we find that by applying weakening
 * and expanding rules, we can reach one of the earlier fpSeqs, that means there is a
     ↪ cycle, and
 * we return cantApply().
 * We do this after trying to discharge closure sequents, because if we can discharge a
     ↪ closure
 * sequent, that also means that there is a cycle, only this cycle is desired.
 *
 * After this, we try applying the ax1 axiom which implicitly also tries to apply
 * weakening and expanding rules to reach just 2 terms upon which it can be applied.
 *
 * If no proof can be found by applying those rules, we try applying the remaining rules,
     ↪ in the
 * order choice, dChoice, concat, test, dTest, and, or, iter, dIter, clo and modm.
 *
 * We saturate the sequent as much as possible, until no more rules can be applied or a
    ↪ cycle is
 * detected. We apply the closure rule last, because we can always move a closure rule
    ↪ application
 * up in a CloG proof with respect to most local rules. Only after saturating the sequent,
    ↪  modm is
 * applied.
 *
 * Different rule orders are possible, however. Shorter or more efficient proofs can
    ↪ sometimes be
 * found by applying the modm rule first, applying the clo rule as early as possible, or
    ↪ trying
 * other rule orders, that might work well on specific sequents.
 *
 * Each rule application returns a MaybeProof, since they recursively call the proofSearch
    ↪ ()
 * algorithm again (except for the tryDisClo() and tryApplyAx1() rules, which call the
 * proofSearchWeakExp() algorithm instead) on the resulting sequent of the application of
    ↪ the
 * corresponding rule.
 *
 * If any rule application return noProof() (except for the closure rule), this function
 * returns noProof() as well. If a rule application returns a cantApply(), the next rule
    ↪ is
 * tried instead. The tryApplyModm() function forms an exception. We try to apply the next
    ↪  rule
 * even if this function returns noProof(), because the modm rule is not exchangeable like
    ↪  the other
 * local rules.
 */
MaybeProof proofSearch(CloGSequent seq, CloSeqs cloSeqs, list[CloGSequent] fpSeqs, int
   ↪ depth) {
        if (depth == 0) return proof(CloGLeaf());

        seq = dup(seq);

        resProof = tryDisClo(seq, cloSeqs);
        if (resProof != noProof()) return resProof;

        if (detectCycles(seq, fpSeqs)) return cantApply();

        resProof = tryApplyAx1(seq);
        if (resProof != cantApply()) return resProof;

        resProof = tryApplyModm(seq, cloSeqs, fpSeqs, depth);
        if (resProof != cantApply() && resProof != noProof()) return resProof;

        resProof = tryApplyClo(seq, cloSeqs, fpSeqs, depth);
        if (resProof != cantApply()) return resProof;

        resProof = tryApplyChoice(seq, cloSeqs, fpSeqs, depth);
        if (resProof != cantApply()) return resProof;

        resProof = tryApplyDChoice(seq, cloSeqs, fpSeqs, depth);
        if (resProof != cantApply()) return resProof;

        resProof = tryApplyConcat(seq, cloSeqs, fpSeqs, depth);
        if (resProof != cantApply()) return resProof;

        resProof = tryApplyTest(seq, cloSeqs, fpSeqs, depth);
        if (resProof != cantApply()) return resProof;

        resProof = tryApplyDTest(seq, cloSeqs, fpSeqs, depth);
        if (resProof != cantApply()) return resProof;

        resProof = tryApplyOr(seq, cloSeqs, fpSeqs, depth);
        if (resProof != cantApply()) return resProof;

        resProof = tryApplyAnd(seq, cloSeqs, fpSeqs, depth);
        if (resProof != cantApply()) return resProof;

        resProof = tryApplyIter(seq, cloSeqs, fpSeqs, depth);
        if (resProof != cantApply()) return resProof;

        return noProof();
}

/*
 * A function that returns whether cycles are detected between the current sequent and the
    ↪  list
 * of saved fixpoint sequents.
 *
 * Input: a current sequent, and a list of fixpoint sequents
 * Output: true, if a cycle is detected, false, otherwise
 *
 * The algorithm loops through all the fixpoint sequents, and if for any of them, a cycle
    ↪ is
 * detected, true is returned. Otherwise, false is returned.
 * A cycle is detected if the current sequent exactly matches any of the sequent saved in
    ↪ the
 * list of saved fixpoint sequents.
 */
bool detectCycles(CloGSequent seq, list[CloGSequent] fpSeqs) {
        for (CloGSequent fpSeq <- fpSeqs)
                if (toSet([<fpSeqTerm.s, toSet(fpSeqTerm.label)> | fpSeqTerm <- fpSeq]) ==
                    ↪ toSet([<seqTerm.s, toSet(seqTerm.label)> | seqTerm <- seq]))
                        return true;

        return false;
}

/*
 * A function that tries discharging a closure sequent on the current sequent.
 *
 * Input: a sequent for which to try to discharge a closure sequent, and the list of
    ↪ currently
 * active closure sequents
 * Output: the proof consisting of possible applications of the weak and exp rules, and
    ↪ the
 * eventual closure sequent discharge, or noProof() if no such proof could be found
 *
 * The algorithm iterates over all the active closure sequents, and for each of them, it
    ↪ checks
 * whether the current sequent has a fixpoint formula corresponding to the closure sequent
    ↪ 's
 * fixpoint formula. If so, and the closure sequent can be reached from the current
    ↪ sequent by
 * applying exp and weak rules, a proof is returned, consisting of these weak and exp
 * applications, and a closure sequent discharge.
 *
 * If for none of the closure sequents, such a fixpoint formula exists, or it can be
    ↪ reached by
 * weak and exp rules applications, noProof() is returned.
 */
MaybeProof tryDisClo(CloGSequent seq, CloSeqs cloSeqs) {
        for (CloGName cn <- cloSeqs) {
                fpSeq = cloSeqs[cn].contextSeq;
                fpIdx = cloSeqs[cn].fpFormulaIdx;
                for (int termIdx <- [0 .. size(seq)]) {
                        if (
                                term(\mod(dIter(Game gamma), GameLog phi), list[CloGName] a,
                                    ↪ _) := seq[termIdx]
                         && term(\mod(dIter(gamma), phi), list[CloGName] b, _) := fpSeq[
                             ↪ fpIdx]
                         && b + cn <= a) {
                                fpSeq[fpIdx] = term(\mod(dIter(gamma), phi), b + cn, false);
                                resProof = proofSearchWeakExp(seq, fpSeq);

                                if (resProof != noProof())
                                        return visit(resProof) {
                                                case CloGUnaryInf(CloGSequent resSeq, weak(),
                                                    ↪ CloGLeaf()) => disClo(resSeq, cn)
                                        };
                        }
                }
        }
        return noProof();
}

/*
 * A function that searches for a proof from one sequent to another, using only the exp
    ↪ and weak
 * rules of CloG.
 *
 * Input: the sequent to start from, the sequent to end up on, and the current depth
 * Output: the proof of the first sequent, which applies exp and weak rules to reach the
    ↪ second
 * sequent, and ends in a (dummy) weakening rule to a CloGLeaf(), or noProof() if the
 * sequent couldn't be reached
 *
 * For each of terms in the first sequent, if it corresponds to a term in the second
     ↪ sequent,
 * we apply the exp rule until it has the same label. If this term does not correspond to
     ↪ a
 * term in the second sequent, we apply weakening to the starting term, and move to the
     ↪ next
 * term. We either end up with the second sequent, in which case a proof is found, or we
     ↪ end up
 * with an empty sequent, when all terms have been removed by weakening, in which case
     ↪ noProof()
 * is returned.
 */
MaybeProof proofSearchWeakExp(CloGSequent seqFrom, CloGSequent seqTo) {
        if (isEmpty(seqFrom))
                return noProof();

        seqFrom = dup(seqFrom);

        if (toSet([<termFrom.s, toSet(termFrom.label)> | termFrom <- seqFrom]) == toSet([<
            ↪ termTo.s, toSet(termTo.label)> | termTo <- seqTo])) {
                return proof(CloGUnaryInf(seqFrom, weak(), CloGLeaf()));
        }

        for (int i <- [0 .. size(seqFrom)]) {
                termFrom = seqFrom[i];
                for (int j <- [0 .. size(seqTo)]) {
                        termTo = seqTo[j];
                        if (termFrom.s == termTo.s && toSet(termFrom.label) > toSet(termTo.
                            ↪ label)) {
                                for (CloGName n <- termFrom.label - termTo.label) {

                                        newSeq = seqFrom;
                                        newSeq[i] = term(termFrom.s, termFrom.label - n,
                                            ↪ false);

                                        subProof = proofSearchWeakExp(newSeq, seqTo);
                                        if (subProof != noProof()) {
                                                seqFrom[i].active = true;
                                                return proof(CloGUnaryInf(seqFrom, exp(),
                                                    ↪ subProof.p));
                                        }
                                }
                        }
                }

                newSeq = delete(seqFrom, i);
                subProof = proofSearchWeakExp(newSeq, seqTo);
                if (subProof != noProof()) {
                        seqFrom[i].active = true;
                        return proof(CloGUnaryInf(seqFrom, weak(), subProof.p));
                }
        }

        return noProof();
}
```

C.2.3 RuleApplications.rsc

```
module ATP::RuleApplications
/*
 * Module providing the functions for each of the rule applications.
 */

import CloG_Base::GLASTs;
import ATP::ATP_Base;
import ATP::ProofSearch;
import Map;
import List;

/*
 * A function applying the "ax1" axiom to a term
 *
 * Input: the sequent to which the "ax1" axiom is applied
 * Output: the sequent resulting from applying the "ax1" axiom
 *
 * For the axiom to be applied, there must be exactly two sequents of the forms
 * "p^[]" and "~p^[]".
 *
 * If these condition are met, an empty sequent (the result of applying the ax1
 * axiom is returned. Otherwise, noSeq() is returned.
 */
MaybeSequent applyAx1(CloGSequent seq) {
       if (size(seq) != 2) return noSeq();

       if (term(atomP(Prop p), [], _) := seq[0] && term(neg(atomP(p)), [], _) := seq[1])
           ↪ {
              return sequent([]);
       }

       if (term(atomP(Prop p), [], _) := seq[1] && term(neg(atomP(p)), [], _) := seq[0])
           ↪ {
              return sequent([]);
       }

       return noSeq();
}

/*
 * A function applying the "ax1" axiom after some potential weakening and
 * expanding of the terms in the sequent.
 *
 * Input: the sequent to which the rule is applied
 * Output: either the CloGProof found after applying weakening/expanding,
 * and the ax1() axiom to the sequent, or noProof() if no such
 * proof could be found
 *
 * If there are two terms "p^a", and "~p^a" in the input sequent, the weak
 * rule is applied to the remaining terms, and the exp rule is used to obtain
 * "p^[]" and "~p^[]". At this point, we can call the applyAx1() function,
 * which will return a leaf.
 *
 * We return the proof that applies weakening and expanding rules, ending in
 * a ax1 application resulting in a CloGLeaf.
 *
 * If no such terms exist, noProof() is returned.
 */
MaybeProof tryApplyAx1(CloGSequent seq) {
        for (int termIdx <- [0 .. size(seq)]) {
                if (term(atomP(Prop p), _, _) := seq[termIdx]) {
                        for (int termIdx2 <- [0 .. size(seq)]) {
                                if (term(neg(atomP(p)), _, _) := seq[termIdx2]) {

                                        CloGSequent weakenTo = termIdx < termIdx2
                                                                     ? [term(atomP(p),
                                                                         ↪ [], true),
                                                                         ↪ term(neg(
                                                                         ↪ atomP(p)),
                                                                         ↪ [], true)]
                                                                     : [term(neg(atomP(p)
                                                                         ↪ ), [], true),
                                                                         ↪  term(atomP(p
                                                                         ↪ ), [], true)
                                                                         ↪ ];

                                        resSeq = applyAx1(weakenTo);
                                        if (resSeq != noSeq()) {
                                                weakenTo[0].active = true;
                                                weakenTo[1].active = true;

                                                resProof = proofSearchWeakExp(seq, weakenTo);

                                                return visit(resProof) {
                                                        case CloGUnaryInf(_, weak(), CloGLeaf()
                                                            ↪ ) => CloGUnaryInf(weakenTo, ax1
                                                            ↪ (), CloGLeaf())
                                                };
                                        }
                                }
                        }
                }
        }

        return cantApply();
}



/* A function applying the "modm" rule to a term
 *
 * Input: the sequent to apply the "modm" rule to
 * Output: the sequent resulting from applying the "modm" rule
 *
 * For the rule to be applied, there must be two terms in the sequent, of forms "<g>phi^a"
    ↪  and
 * "<g^d>psi^b"
 *
 * If these conditions are met, a sequent containing "phi^a" and "psi^b" is returned.
    ↪ Otherwise,
 * noSeq() is returned.
 */
MaybeSequent applyModm(CloGSequent seq) {

       if (size(seq) != 2)
              return noSeq();

       if (term(\mod(Game g, GameLog phi), list[CloGName] a, _) := seq[0] && term(\mod(
           ↪ dual(g), GameLog psi), list[CloGName] b, _) := seq[1]) {
              return sequent([term(phi, a, false), term(psi, b, false)]);
       }
       if (term(\mod(Game g, GameLog phi), list[CloGName] a, _) := seq[1] && term(\mod(
           ↪ dual(g), GameLog psi), list[CloGName] b, _) := seq[0]) {
              return sequent([term(psi, b, false), term(phi, a, false)]);
       }

       return noSeq();
}

/*
 * All of the remaining tryApply<rulename>() functions defined in here have the
 * following inputs' and outputs:
 *
 * Input: the sequent to which the rule is applied, the map of closure
 * sequents, the list of fixpoint sequents, and the current depth
 * Output: either the CloGProof found after applying the rule to the sequent,
 * or noProof() if no such proof could be found
 */

/*
 * A function applying the "modm" rule to a sequent, after some potential
 * weakening of the terms in the given sequent. The main proofSearch()
 * algorithm is called on the sequent resulting from applying this "modm" rule.
 *
 * For the rule to be applied, there must be 2 terms in the sequent of the
 * form "<g>phi^[a]", and "~<g^d>psi^[b]". The remaining terms are removed by
 * weakening, and a proof search is done on the resulting sequent.
 *
 * If a subproof is found for this resulting sequent, we return the proof
 * that includes the weakening rule applications and ends in this subproof.
 *
 * The rule application is tried for any 2 terms of the form. If none of these
 * return a proof, or there are no 2 terms of the appropriate form, noProof()
 * is returned.
 */
```

```
MaybeProof tryApplyModm(CloGSequent seq, CloSeqs cloSeqs, list[CloGSequent] fpSeqs, int
    ↪ depth) {
        for (int termIdx <- [0 .. size(seq)]) {
                if (term(\mod(Game g, GameLog phi), list[CloGName] a, _) := seq[termIdx]) {
                        for (int termIdx2 <- [0 .. size(seq)]) {
                                if (term(\mod(dual(g), GameLog psi), list[CloGName] b, _) :=
                                    ↪ seq[termIdx2]) {

                                        CloGSequent weakenTo = termIdx < termIdx2
                                                                        ? [term(\mod(g, phi)
                                                                            ↪ , a, false),
                                                                            ↪ term(\mod(
                                                                            ↪ dual(g), psi)
                                                                            ↪ , b, false)]
                                                                        : [term(\mod(dual(g)
                                                                            ↪ , psi), b,
                                                                            ↪ false), term
                                                                            ↪ (\mod(g, phi)
                                                                            ↪ , a, false)];

                                        MaybeSequent resSeq = applyModm(weakenTo);

                                        if (resSeq != noSeq()) {
                                                subProof = proofSearch(resSeq.seq, cloSeqs,
                                                    ↪ fpSeqs, depth - 1);
                                                if (subProof != noProof() && subProof !=
                                                    ↪ cantApply()) {
                                                        weakenTo[0].active = true;
                                                        weakenTo[1].active = true;

                                                        resProof = proofSearchWeakExp(seq,
                                                            ↪ weakenTo);

                                                        return visit(resProof) {
                                                                case CloGUnaryInf(_, weak(),
                                                                    ↪ CloGLeaf()) =>
                                                                    ↪ CloGUnaryInf(weakenTo,
                                                                    ↪ modm(), subProof.p)
                                                        };
                                                }
                                        }
                                }
                        }
                }
        }

        return noProof();
}

/*
 * A function applying the "and" rule to a term
 *
 * Input: the sequent and the index of the term therein to which to apply the "and" rule
 * Output: the pair of sequents resulting from applying the "and" rule
 *
 * For the rule to be applied, the term at the specified index must be of
 * the form "(phi & psi)^a".
 *
 * If this condition is met, a pair of sequents is returned, such that for one of
 * them, the specified term is replaced by "phi^a", and for the other, it is
 * replaced by "psi^a". Otherwise, noSeqs() is returned.
 */
MaybeSequents applyAnd(CloGSequent seq, int termIdx) {
        if (term(and(GameLog phi, GameLog psi), list[CloGName] a, _) := seq[termIdx]) {
                copySeq = seq;
                seq[termIdx] = term(phi, a, false);
                copySeq[termIdx] = term(psi, a, false);

                return sequents(seq, copySeq);
        }
        return noSeqs();
}

/*
 * A function applying the "and" rule to a sequent and calling the main proof search
 * algorithm on the resulting sequents.
 *
 * For any of the terms in the sequent, the algorithm checks tries to apply the "and"
 * rule. For the first successful application, a proof search is done on the resulting
    ↪ sequent.
 * If either of these proof searches returns a noProof() or cantApply(), that return value
 * is propagated. If two subproofs are found, A CloGBinaryInf with the resulting subproofs
    ↪ ,
 * is returned.
 *
 * If the "and" rule could not be applied to any term in the sequent, cantApply() is
    ↪ returned.
 */
MaybeProof tryApplyAnd(CloGSequent seq, CloSeqs cloSeqs, list[CloGSequent] fpSeqs, int
    ↪ depth) {
        for (int termIdx <- [0 .. size(seq)]) {
                MaybeSequents resSeqs = applyAnd(seq, termIdx);
                if (resSeqs != noSeqs()) {
                        MaybeProof subProofL = proofSearch(resSeqs.left, cloSeqs, fpSeqs,
                            ↪ depth - 1);

                        if (subProofL == noProof() ||subProofL == cantApply())
                                return subProofL;

                        MaybeProof subProofR = proofSearch(resSeqs.right, cloSeqs, fpSeqs,
                            ↪ depth - 1);

                        if (subProofR == noProof() ||subProofR == cantApply())
                                return subProofR;

                        seq[termIdx].active = true;
                        return proof(CloGBinaryInf(seq, subProofL.p, subProofR.p));
                }
        }

        return cantApply();
}

/*
 * A function applying the "or" rule to a term
 *
 * Input: the sequent and the index of the term therein to which to apply the "or" rule
 * Output: the sequent resulting from applying the "or" rule
 *
 * For the rule to be applied, the term at the specified index must be of
 * the form "(phi | psi)^a".
 *
 * If this condition is met, the specified term in the given sequent is replaced by two
 * terms "phi^a", and "psi^a" and the sequent is returned. Otherwise, noSeq() is returned.
 */
MaybeSequent applyOr(CloGSequent seq, int termIdx) {
        if (term(or(GameLog phi, GameLog psi), list[CloGName] a, _) := seq[termIdx]) {
                return sequent(seq[0 .. termIdx] + term(phi, a, false) + term(psi, a, false
                    ↪ ) + seq[termIdx+1 .. size(seq)]);
        }
        return noSeq();
}

/*
 * A function applying the "or" rule to a sequent and calling the main
 * proof search algorithm on the resulting sequent.
 *
 * For any of the terms in the sequent, the algorithm checks tries to apply the "or"
 * rule. For the first successful application, a proof search is done on the resulting
    ↪ sequent.
 * If a subproof is found, A CloGUnaryInf with the resulting subproof, and the applied orR
    ↪ ()
 * rule is returned. Otherwise, the subProof itself is returned, which can be noProof()
 * or cantApply().
 * If the "or" rule could not be applied to any term in the sequent, cantApply() is
    ↪ returned.
 */
MaybeProof tryApplyOr(CloGSequent seq, CloSeqs cloSeqs, list[CloGSequent] fpSeqs, int
    ↪ depth) {
        for (int termIdx <- [0 .. size(seq)]) {
                MaybeSequent resSeq = applyOr(seq, termIdx);
                if (resSeq != noSeq()) {
                        subProof = proofSearch(resSeq.seq, cloSeqs, fpSeqs, depth - 1);
                        if (subProof != noProof() && subProof != cantApply()) {
                                seq[termIdx].active = true;
                                return proof(CloGUnaryInf(seq, orR(), subProof.p));
                        }
                        return subProof;
                }
        }

        return cantApply();
```

```
}

/*
 * A function applying the "choice" rule to a term
 *
 * Input: the sequent and the index of the term therein to which to apply the "choice"
     ↪ rule
 * Output: the sequent resulting from applying the "choice" rule
 *
 * For the rule to be applied, the term at the specified index must be of
 * the form "(<gamma || delta>phi)^a".
 *
 * If this condition is met, the specified term in the given sequent is replaced by
 * the term "(<gamma>phi | <delta> phi)^a" and the sequent is returned. Otherwise,
 * noSeq() is returned.
 */
MaybeSequent applyChoice(CloGSequent seq, int termIdx) {
        if (term(\mod(choice(Game gamma, Game delta), GameLog phi), list[CloGName] a, _)
            ↪ := seq[termIdx]) {
                seq[termIdx] = term(or(\mod(gamma, phi), \mod(delta, phi)), a, false);
                return sequent(seq);
        }
        return noSeq();
}

/*
 * A function applying the "choice" rule to a sequent and calling the main
 * proof search algorithm on the resulting sequent.
 *
 * For any of the terms in the sequent, the algorithm tries to apply the "choice"
 * rule. For the first successful application, a proof search is done on the resulting
     ↪ sequent.
 * If a subproof is found, A CloGUnaryInf with the resulting subproof, and the applied
     ↪ choiceR()
 * rule is returned. Otherwise, the subProof itself is returned, which can be noProof()
 * or cantApply().
 * If the "choice" rule could not be applied to any term in the sequent, cantApply() is
     ↪ returned.
 */
MaybeProof tryApplyChoice(CloGSequent seq, CloSeqs cloSeqs, list[CloGSequent] fpSeqs, int
    ↪  depth) {
        for (int termIdx <- [0 .. size(seq)]) {
                MaybeSequent resSeq = applyChoice(seq, termIdx);
                if (resSeq != noSeq()) {
                        MaybeProof subProof = proofSearch(resSeq.seq, cloSeqs, fpSeqs, depth
                            ↪  - 1);
                        if (subProof != noProof() && subProof != cantApply()) {
                                seq[termIdx].active = true;
                                return proof(CloGUnaryInf(seq, choiceR(), subProof.p));
                        }
                        return subProof;
                }
        }
```

```
       return cantApply();
}

/*
 * A function applying the "dChoice" rule to a term
 *
 * Input: the sequent and the index of the term therein to which to apply the "dChoice"
    ↪ rule
 * Output: the sequent resulting from applying the "dChoice" rule
 *
 * For the rule to be applied, the term at the specified index must be of
 * the form "(<gamma && delta>phi)^a".
 *
 * If this condition is met, the specified term in the given sequent is replaced by
 * the term "(<gamma>phi & <delta> phi)^a" and the sequent is returned. Otherwise,
 * noSeq() is returned.
 */
MaybeSequent applyDChoice(CloGSequent seq, int termIdx) {
       if (term(\mod(dChoice(Game gamma, Game delta), GameLog phi), list[CloGName] a, _)
           ↪ := seq[termIdx]) {
              seq[termIdx] = term(and(\mod(gamma, phi), \mod(delta, phi)), a, false);
              return sequent(seq);
       }
       return noSeq();
}

/*
 * A function applying the "dChoice" rule to a sequent and calling the main
 * proof search algorithm on the resulting sequent.
 *
 * For any of the terms in the sequent, the algorithm tries to apply the "dChoice"
 * rule. For the first successful application, a proof search is done on the resulting
    ↪ sequent.
 * If a subproof is found, A CloGUnaryInf with the resulting subproof, and the applied
    ↪ dChoiceR()
 * rule is returned. Otherwise, the subProof itself is returned, which can be noProof()
 * or cantApply().
 * If the "dChoice" rule could not be applied to any term in the sequent, cantApply() is
    ↪ returned.
 */
MaybeProof tryApplyDChoice(CloGSequent seq, CloSeqs cloSeqs, list[CloGSequent] fpSeqs,
    ↪ int depth) {
       for (int termIdx <- [0 .. size(seq)]) {
              MaybeSequent resSeq = applyDChoice(seq, termIdx);
              if (resSeq != noSeq()) {
                     MaybeProof subProof = proofSearch(resSeq.seq, cloSeqs, fpSeqs, depth
                         ↪  - 1);
                     if (subProof != noProof() && subProof != cantApply()) {
                            seq[termIdx].active = true;
                            return proof(CloGUnaryInf(seq, dChoiceR(), subProof.p));
                     }
                     return subProof;
              }
       }
```

```
      return cantApply();
}


/*
 * A function applying the "concat" rule to a term
 *
 * Input: the sequent and the index of the term therein to which to apply the "concat" rule
 * Output: the sequent resulting from applying the "concat" rule
 *
 * For the rule to be applied, the term at the specified index must be of
 * the form "(<gamma; delta>phi)^a".
 *
 * If this condition is met, the specified term in the given sequent is replaced by
 * the term "(<gamma><delta>phi)^a" and the sequent is returned. Otherwise, noSeq()
 * is returned.
 */
MaybeSequent applyConcat(CloGSequent seq, int termIdx) {
      if (term(\mod(concat(Game gamma, Game delta), GameLog phi), list[CloGName] a, _) := seq[termIdx]) {
            seq[termIdx] = term(\mod(gamma, \mod(delta, phi)), a, false);
            return sequent(seq);
      }
      return noSeq();
}

/*
 * A function applying the "concat" rule to a sequent and calling the main
 * proof search algorithm on the resulting sequent.
 *
 * For any of the terms in the sequent, the algorithm tries to apply the "concat"
 * rule. For the first successful application, a proof search is done on the resulting sequent.
 * If a subproof is found, A CloGUnaryInf with the resulting subproof, and the applied concatR()
 * rule is returned. Otherwise, the subProof itself is returned, which can be noProof()
 * or cantApply().
 * If the "concat" rule could not be applied to any term in the sequent, cantApply() is returned.
 */
MaybeProof tryApplyConcat(CloGSequent seq, CloSeqs cloSeqs, list[CloGSequent] fpSeqs, int depth) {
      for (int termIdx <- [0 .. size(seq)]) {
            MaybeSequent resSeq = applyConcat(seq, termIdx);
            if (resSeq != noSeq()) {
                  MaybeProof subProof = proofSearch(resSeq.seq, cloSeqs, fpSeqs, depth - 1);
                  if (subProof != noProof() && subProof != cantApply()) {
                        seq[termIdx].active = true;
                        return proof(CloGUnaryInf(seq, concatR(), subProof.p));
                  }
                  return subProof;
```

```
                }
        }

        return cantApply();
}

/*
 * A function applying the "test" rule to a term
 *
 * Input: the sequent and the index of the term therein to which to apply the "test" rule
 * Output: the sequent resulting from applying the "test" rule
 *
 * For the rule to be applied, the term at the specified index must be of
 * the form "(<psi?>phi)^a".
 *
 * If this condition is met, the specified term in the given sequent is replaced by
 * the term "(psi & phi)^a" and the sequent is returned. Otherwise, noSeq() is returned.
 */
MaybeSequent applyTest(CloGSequent seq, int termIdx) {
        if (term(\mod(\test(GameLog psi), GameLog phi), list[CloGName] a, _) := seq[
            ↪ termIdx]) {
                seq[termIdx] = term(and(psi, phi), a, false);
                return sequent(seq);
        }
        return noSeq();
}

/*
 * A function applying the "test" rule to a sequent and calling the main
 * proof search algorithm on the resulting sequent.
 *
 * For any of the terms in the sequent, the algorithm tries to apply the "test"
 * rule. For the first successful application, a proof search is done on the resulting
     ↪ sequent.
 * If a subproof is found, A CloGUnaryInf with the resulting subproof, and the applied
     ↪ testR()
 * rule is returned. Otherwise, the subProof itself is returned, which can be noProof()
 * or cantApply().
 * If the "test" rule could not be applied to any term in the sequent, cantApply() is
     ↪ returned.
 */
MaybeProof tryApplyTest(CloGSequent seq, CloSeqs cloSeqs, list[CloGSequent] fpSeqs, int
    ↪ depth) {
        for (int termIdx <- [0 .. size(seq)]) {

                MaybeSequent resSeq = applyTest(seq, termIdx);
                if (resSeq != noSeq()) {
                        MaybeProof subProof = proofSearch(resSeq.seq, cloSeqs, fpSeqs, depth
                            ↪  - 1);
                        if (subProof != noProof() && subProof != cantApply()) {
                                seq[termIdx].active = true;
                                return proof(CloGUnaryInf(seq, testR(), subProof.p));
                        }
                        return subProof;
            }
      }

      return cantApply();
}


/*
 * A function applying the "dTest" rule to a term
 *
 * Input: the sequent and the index of the term therein to which to apply the "dTest" rule
 * Output: the sequent resulting from applying the "dTest" rule
 *
 * For the rule to be applied, the term at the specified index must be of
 * the form "(<psi!>phi)^a".
 *
 * If this condition is met, the specified term in the given sequent is replaced by
 * the term "(psi | phi)^a" and the sequent is returned. Otherwise, noSeq() is returned.
 */
MaybeSequent applyDTest(CloGSequent seq, int termIdx) {
      if (term(\mod(dTest(GameLog psi), GameLog phi), list[CloGName] a, _) := seq[
         ↪ termIdx]) {
            seq[termIdx] = term(or(psi, phi), a, false);
            return sequent(seq);
      }
      return noSeq();
}

/*
 * A function applying the "dTest" rule to a sequent and calling the main
 * proof search algorithm on the resulting sequent.
 *
 * For any of the terms in the sequent, the algorithm tries to apply the "dTest"
 * rule. For the first successful application, a proof search is done on the resulting
    ↪ sequent.
 * If a subproof is found, A CloGUnaryInf with the resulting subproof, and the applied
    ↪ dTestR()
 * rule is returned. Otherwise, the subProof itself is returned, which can be noProof()
 * or cantApply().
 * If the "dTest" rule could not be applied to any term in the sequent, cantApply() is
    ↪ returned
 */
MaybeProof tryApplyDTest(CloGSequent seq, CloSeqs closeqs, list[CloGSequent] fpSeqs, int
   ↪ depth) {
      for (int termIdx <- [0 .. size(seq)]) {

            MaybeSequent resSeq = applyDTest(seq, termIdx);
            if (resSeq != noSeq()) {
                  MaybeProof subProof = proofSearch(resSeq.seq, cloSeqs, fpSeqs, depth
                     ↪  - 1);
                  if (subProof != noProof() && subProof != cantApply()) {
                        seq[termIdx].active = true;
                        return proof(CloGUnaryInf(seq, dTestR(), subProof.p));
                  }
```

```
                        return subProof;
                }
        }

        return cantApply();
}

/*
 * A function applying the "iter" rule to a term
 *
 * Input: the sequent and the index of the term therein to which to apply the "iter" rule,
 * and the list of closure sequents
 * Output: the sequent resulting from applying the "iter" rule
 *
 * For the rule to be applied, the term at the specified index must be of
 * the form "(<gamma*>phi)^a".
 * Each of the names in the label "a" must be smaller than or equal to this
 * "<gamma*>phi" fixpoint formula, according to the order on fixpoint formulae
 * defined in the literature.
 *
 * If these conditions are met, the specified term in the given sequent is replaced by
 * the term "(phi | <gamma><gamma*>phi)^a" and the sequent is returned. Otherwise,
 * noSeq() is returned.
 */
MaybeSequent applyIter(CloGSequent seq, int termIdx, CloSeqs cloSeqs) {
        if (term(\mod(iter(Game gamma), GameLog phi), list[CloGName] a, _) := seq[termIdx
            ↪ ]) {
                for (CloGName x <- a)
                        if (!fpLessThanOrEqualTo(cloSeqs[x].contextSeq[cloSeqs[x].
                            ↪ fpFormulaIdx].s, \mod(iter(gamma), phi)))
                                return noSeq();

                seq[termIdx] = term(or(phi, \mod(gamma, \mod(iter(gamma), phi))), a, false)
                    ↪ ;
                return sequent(seq);
        }
        return noSeq();
}

/*
 * A function applying the "iter" rule to a sequent and calling the main
 * proof search algorithm on the resulting sequent.
 *
 * For any of the terms in the sequent, the algorithm tries to apply the "iter"
 * rule. For the first successful application, a proof search is done on the resulting
     ↪ sequent.
 * If a subproof is found, the current sequent is added to the list of fixpoint sequents,
 * and a CloGUnaryInf with the resulting subproof, and the applied iterR() rule is
     ↪ returned.
 * Otherwise, the subProof itself is returned, which can be noProof() or cantApply().
 * If the "iter" rule could not be applied to any term in the sequent, cantApply() is
     ↪ returned.
 */
```

```
MaybeProof tryApplyIter(CloGSequent seq, CloSeqs cloSeqs, list[CloGSequent] fpSeqs, int
    ↪ depth) {
        for (int termIdx <- [0 .. size(seq)]) {

                MaybeSequent resSeq = applyIter(seq, termIdx, cloSeqs);

                if (resSeq != noSeq()) {
                        fpSeqs += [seq];
                        MaybeProof subProof = proofSearch(resSeq.seq, cloSeqs, fpSeqs, depth
                            ↪  - 1);
                        if (subProof != noProof() && subProof != cantApply()) {
                                seq[termIdx].active = true;
                                return proof(CloGUnaryInf(seq, iterR(), subProof.p));
                        }
                        return subProof;
                }
        }
        return cantApply();
}

/*
 * A function applying the "clo" rule to a term
 *
 * Input: the sequent and the index of the term therein to which to apply the "clo" rule,
 * and the list of closure sequents
 * Output: a tuple containing the sequent resulting from applying the clo rule, and
 * the new name associated with the fixpoint formula of the term that the rule
 * was applied to
 *
 * For the rule to be applied, the term at the specified index must be of
 * the form "(<gamma^x>phi)^a".
 * Each of the names in the label "a" must be smaller than or equal to this
 * "(<gamma^x>phi)^a" fixpoint formula, according to the order on fixpoint formulae
 * defined in the literature.
 *
 * If these conditions are met, a new name is created, x_n, where n is the
 * number of closure sequents (or closure sequents names, named x_0 to x_{n-1})
 * currently in the list of closure sequents, and the specified term in the given
 * sequent is replaced by the term "(phi & <gamma><gamma^x>phi)^{a + x_n}" and a
 * tuple of this sequent and the new associated name is returned. Otherwise, a tuple
 * of noSeq() and an empty name is returned.
 */
tuple[MaybeSequent, CloGName] applyClo(CloGSequent seq, int termIdx, CloSeqs cloSeqs) {
        if (term(\mod(dIter(Game gamma), GameLog phi), list[CloGName] a, _) := seq[termIdx
            ↪ ]) {

                for (CloGName x <- a) {
                        GameLog cloForm = cloSeqs[x].contextSeq[cloSeqs[x].fpFormulaIdx].s;
                        if (!fpLessThanOrEqualTo(cloForm, \mod(dIter(gamma), phi)))
                                return <noSeq(), name("")>;
                }

                CloGName newName = nameS("x", size(cloSeqs));
                seq[termIdx] = term(and(phi, \mod(gamma, \mod(dIter(gamma), phi))), a +
                    ↪ newName, false);
                return <sequent(seq), newName>;
        }
        return <noSeq(), name("")>;
}

/*
 * A function applying the "clo" rule to a sequent and calling the main
 * proof search algorithm on the resulting sequent.
 *
 * For any of the terms in the sequent, the algorithm tries to apply the "clo"
 * rule. For all successful applications, a proof search is done on the resulting sequent.
 * If a subproof is found, the current sequent is added to the list of fixpoint sequents,
 * and the new name associated with the closure rule application is added to the closure
 * sequents as the key to the associated context sequence and the index of the term in the
 * sequent that contains the relevant fixpoint formula.
 *
 * If a subproof is found, a CloGUnaryInf with the resulting subproof, and the applied clo
     ↪ ()
 * rule (with the new name) is returned.
 * If the "clo" rule could not be applied to any term in the sequent, cantApply() is
     ↪ returned.
 * If the "clo" rule could be applied to a term in the sequent, but no subproof was found,
 * noProof() is returned.
 */
MaybeProof tryApplyClo(CloGSequent seq, CloSeqs cloSeqs, list[CloGSequent] fpSeqs, int
    ↪ depth) {
        bool canApply = false;

        for (int termIdx <- [0 .. size(seq)]) {

                tuple[MaybeSequent resSeq, CloGName newName] res = applyClo(seq, termIdx,
                    ↪ cloSeqs);
                if (res.resSeq != noSeq()) {
                        canApply = true;

                        cloSeqs += (res.newName: <seq, termIdx>);
                        fpSeqs += [seq];
                        MaybeProof subProof = proofSearch(res.resSeq.seq, cloSeqs, fpSeqs,
                            ↪ depth - 1);
                        if (subProof != noProof() && subProof != cantApply()) {
                                seq[termIdx].active = true;
                                return proof(CloGUnaryInf(seq, clo(res.newName), subProof.p))
                                    ↪ ;
                        }
                }
        }
        return canApply ? noProof() : cantApply();
}
```

C.2.4 PostProcess.rsc

```
module ATP::PostProcess
/*
 * A module that deals with the post-processing of proofs
 * obtained from the Automated Theorem Prover
 */

import CloG_Base::GLASTs;
import ATP::ATP_Base;

/*
 * A function that replaces closure rules that no not have
 * a discharged assumption by a dIter rule instead.
 *
 * Input: A CloGProof that may contain closure rule applications
 * without a discharged assumption appearing somewhere above
 * the proof node at which the closure rule was applied
 * Output: A CloGProof with its closure rule applications without
 * assumptions replaced by dIter applications
 *
 * For each of the nodes at which a closure rule was applied, and
 * above which no discharged assumption appears, the replaceCloAt()
 * function is called, which handles the actual replacement.
 */
CloGProof replaceUnusedClos(CloGProof proof) {
        return visit(proof) {
                case subProof:CloGUnaryInf(_, clo(CloGName n), CloGProof inf): {
                        if (/disClo(_, n) !:= inf) {
                                insert replaceCloAt(subProof, n);
                        }
                }
        }
}

/*
 * A function that replaces a specific closure rule application
 * in a proof by a dIter rule application, and updates the rest
 * of the proof accordingly
 *
 * Input: A CloGProof with a closure rule application at the root,
 * and the name associated with this closure rule application
 * Output: A CloGProof with the closure rule application at the root
 * replaced by a dIter rule application, and no more closure
 * rule name appearing in sequents above this proof node
 *
 * Replaces the rule at the root by a dIter rule, then visits the
 * rest of the proof, and for all sequents above the starting node,
 * the name associated with the original closure rule application
 * is removed.
 */
CloGProof replaceCloAt(CloGProof proof, CloGName name) {
        proof.rule = dIterR();
        return visit(proof) {
                case subP:CloGUnaryInf(CloGSequent seq, _, _): {
                        CloGSequent newSeq = [];
                        for (term <- seq) {
                                if (t:term(_, list[CloGName] label, _) := term) {
                                        t.label = label - name;
                                        newSeq += t;
                                }
                        }
                        subP.seq = newSeq;
                        insert subP;
                }
                case subP:CloGBinaryInf(CloGSequent seq, _, _): {
                        CloGSequent newSeq = [];
                        for (term <- seq) {
                                if (t:term(_, list[CloGName] label, _) := term) {
                                        t.label = label - name;
                                        newSeq += t;
                                }
                        }
                        subP.seq = newSeq;
                        insert subP;
                }
                case subP:disClo(CloGSequent seq, _): {
                        CloGSequent newSeq = [];
                        for (term <- seq) {
                                if (t:term(_, list[CloGName] label, _) := term) {
                                        t.label = label - name;
                                        newSeq += t;
                                }
                        }
                        subP.seq = newSeq;
                        insert subP;
                }
        }
}
```

C.2.5 CloG_ATP_Tool.rsc

```
module ATP::CloG_ATP_Tool
/*
 * Main module for the automated theorem prover tool
 */

import CloG_Base::CloGSyntax;
import CloG_Base::GLASTs;
import CloG_Base::CST2AST_CloG;
import CloG_Base::LaTeXOutput;
import ATP::ATP_Base;
import ATP::ProofSearch;
import ATP::PostProcess;

import util::IDE;
import ParseTree;
import IO;

// Call IDE() in terminal to activate parse checker and keyword highlighting for .seq
    ↪ files in Eclipse.
void IDE() {
       start[SCloGSequent] clogSeq(str src, loc l) {
              return parse(#start[SCloGSequent], src, l);
       }

       registerLanguage("CloGSeq", "seq", clogSeq);
}

// Main function is split up for modularity and to help with testing in the terminal.

// Get the location of an input .seq file.
loc inputLoc(str file) {
       return (|project://CloG-ATP/input| + file)[extension=".seq"];
}

// Form the location for an output .tex file
loc outputLoc(str file) {
       return (|project://CloG-ATP/output| + file)[extension=".tex"];
}

// Parse input .seq file and output abstract syntax tree for the CloG Sequent
CloGSequent getCloGAST(str file){
       loc l = inputLoc(file);
       start[SCloGSequent] cst = parse(#start[SCloGSequent], l);
       return cst2astCloG(cst);
}

// Input abstract syntax tree for CloG proof and output .tex proof tree to the given
    ↪ output file
void CloG2LaTeX(CloGProof p, str out){
       loc l = outputLoc(out);
       LaTeXOutput(p, l);
}

// Display the sequent at file location "in" as a CloG proof tree as a .tex proof tree
    ↪ displaying the
// sequent and a dummy "weak" rule application.
void input2latex(str \in, str out){
       CloG2LaTeX(CloGUnaryInf(getCloGAST(\in), weak(), CloGLeaf()), out);
}

// Call main proofSearch_Tool function with an empty list of closure sequent and fixpoint
    ↪  sequent files
// and a depth of -1.
void proofSearch_Tool(str file) {
       proofSearch_Tool(file, [], [], -1);
}
```

```
// Call main proofSearch_Tool function with an empty list of closure sequent and fixpoint
    ↪  sequent files.
void proofSearch_Tool(str file, int depth) {
       proofSearch_Tool(file, [], [], depth);
}

// Call main proofSearch_Tool function with a depth of -1.
void proofSearch_Tool(str file, list[tuple[str, int]] cloSeqFiles, list[str] fpSeqsFiles)
    ↪  {
       proofSearch_Tool(file, cloSeqFiles, fpSeqsFiles, -1);
}

// Input .seq file name, do a proof search on the sequent with a given list of names to
    ↪ files with closure
// sequents and integers corresponding to the relevant fixpoint formula within those
    ↪ sequents, a given
// list of fixpoint sequents in a similar manner (but without the integer), and a maximum
    ↪  proof search depth.
// Replace unused closure rule applications and display the result in LaTeX
// (or display "fail!" if no proof could be found).
void proofSearch_Tool(str file, list[tuple[str, int]] cloSeqFiles, list[str] fpSeqsFiles,
    ↪  int depth){
       CloGSequent seqAST = getCloGAST(file);

       CloSeqs cloSeqs = ();
       for (int i <- [0 .. size(cloSeqFiles)])
              cloSeqs += (nameS("x", i): <getCloGAST(cloSeqFiles[i][0]), cloSeqFiles[i
                  ↪ ][1]>);

       list[CloGSequent] fpSeqs = [];
       for (str fpSeqFile <- fpSeqsFiles)
              fpSeqs += [getCloGAST(fpSeqFile)];

       MaybeProof resProof = proofSearch(seqAST, cloSeqs, fpSeqs, depth);
       if (resProof != noProof()) {
              CloGProof validProof = replaceUnusedClos(resProof.p);
              LaTeXOutput(validProof, outputLoc(file));
       } else
              println("fail!\n");
}
```

C.3 integrationTests

C.3.1 Proof_tests.rsc

```
module integrationTests::Proof_tests
/*
 * Module to execute the integration tests
 */

import ATP::CloG_ATP_Tool;
```

```
/*
 * A function that generates LaTeX proofs for each of the sequents in the input folder (
    ↪ manually)
 */
void executeTests() {
        proofSearch_Tool("01 (empty)");
        proofSearch_Tool("02 (single)");
        proofSearch_Tool("03 (single named)");
        proofSearch_Tool("04 (double)");
        proofSearch_Tool("05 (double named)");
        proofSearch_Tool("06 (ax1)");
        proofSearch_Tool("07 (ax1 with labels)");
        proofSearch_Tool("08 (ax1 with side formulae)");
        proofSearch_Tool("09 (ax1 with side formulae and labels)");

        proofSearch_Tool("10 (disClo)/seq", [<"10 (disClo)/cloSeq and fpSeq", 0>], ["10 (
            ↪ disClo)/cloSeq and fpSeq"]);

        proofSearch_Tool("11 (or)");
        proofSearch_Tool("12 (or with side formulae)");
        proofSearch_Tool("13 (and)");
        proofSearch_Tool("14 (modm)");
        proofSearch_Tool("15 (choice)");
        proofSearch_Tool("16 (dChoice)");
        proofSearch_Tool("17 (concat)");
        proofSearch_Tool("18 (dTest)");
        proofSearch_Tool("19 (test)");
        proofSearch_Tool("20 (monotonicity)");
        proofSearch_Tool("21 (iter 1)");
        proofSearch_Tool("22 (iter 2)");
        proofSearch_Tool("23 (iter 3)");
        proofSearch_Tool("24 (iter 4)");
        proofSearch_Tool("25 (dIter & iter)");
        proofSearch_Tool("26 (dIter & iter 2)");
        proofSearch_Tool("27 (double dIter & iter)");
        proofSearch_Tool("28 (multi iter)");
        proofSearch_Tool("29 (multi iter 2)");
        proofSearch_Tool("30 (multi dIter)");
        proofSearch_Tool("31 (double iter fail)");
        proofSearch_Tool("32 (double dIter fail)");
        proofSearch_Tool("33 (multi iter fail)");
        proofSearch_Tool("34 (multi dIter fail)");
        proofSearch_Tool("35 (iter dIter alternate fail)");
        proofSearch_Tool("36 (bigSeq1)");
        proofSearch_Tool("37 (bigSeq2)");
        proofSearch_Tool("38 (proof 1)");
        proofSearch_Tool("39 (proof 2)");
}
```

C.4 unitTests

C.4.1 ATP_Base_test.rsc

```
module unitTests::ATP_Base_test
/*
 * Module to unit test the functions in the ATP::ATP_Base module, namely the
 * subTerm() and fpLessThanOrEqualTo() functions.
 */

import ATP::ATP_Base;
import CloG_Base::GLASTs;
import Exception;

// Subterm

// A game should be a subterm of itself
// (a <= a)
test bool subTerm_test_1() =
       subTerm(atomG(agame("a")), atomG(agame("a"))) == true;
// A game should be a subterm of a unary operator applied to itself
// (a <= a^d)
test bool subTerm_test_2() =
       subTerm(atomG(agame("a")), dual(atomG(agame("a")))) == true;
// A game should be a subterm of a binary operator applied to itself and something else
// (a <= a && b)
test bool subTerm_test_3() =
       subTerm(atomG(agame("a")), dChoice(atomG(agame("a")), atomG(agame("b")))) == true;
// This should also work if the game appears not at the start of the other formula
// (a <= b && a)
test bool subTerm_test_4() =
       subTerm(atomG(agame("a")), dChoice(atomG(agame("b")), atomG(agame("a")))) == true;
// Further nesting
// (a <= (b && a)*)
test bool subTerm_test_5() =
       subTerm(atomG(agame("a")), iter(dChoice(atomG(agame("b")), atomG(agame("a"))))) ==
           ↪  true;
// A game should be a subterm a test of a game logic formula containing that game
// (a <= <a>p?)
test bool subTerm_test_6() =
       subTerm(atomG(agame("a")), \test(\mod(atomG(agame("a")), atomP(prop("p"))))) ==
           ↪ true;
// A composite game can also be a subterm of a bigger composite game
// ((a && b) <= (a && b)*)
test bool subTerm_test_7() =
       subTerm(dChoice(atomG(agame("a")), atomG(agame("b"))), iter(dChoice(atomG(agame("a
           ↪ ")), atomG(agame("b"))))) == true;
// A simple game should not be a subterm of a different simple game
// (a !<= b)
test bool subTerm_test_8() =
       subTerm(atomG(agame("a")), atomG(agame("b"))) == false;
// A composite game should not be a subterm of a simple game
// (a^d !<= a)
test bool subTerm_test_9() =
       subTerm(dual(atomG(agame("a"))), atomG(agame("a"))) == false;
// A game should be fully contained to be a subterm of another game
// (a && a !<= a && b)
test bool subTerm_test_10() =
       subTerm(dChoice(atomG(agame("a")), atomG(agame("a"))), dChoice(atomG(agame("a")),
           ↪ atomG(agame("b")))) == false;
// Order matters
// (a && b !<= b && a)
test bool subTerm_test_11() =
       subTerm(dChoice(atomG(agame("a")), atomG(agame("b"))), dChoice(atomG(agame("b")),
           ↪ atomG(agame("a")))) == false;
// Order matters
// (a && b !<= a* && b)
test bool subTerm_test_12() =
       subTerm(dChoice(atomG(agame("a")), atomG(agame("b"))), dChoice(iter(atomG(agame("a
           ↪ "))), atomG(agame("b")))) == false;



// Less Than Or Equal To (Fixpoint Formula Ordering)

// <a*>p <= <a*>q
test bool fpLessThanOrEqualTo_test_1() =
       fpLessThanOrEqualTo(\mod(iter(atomG(agame("a"))), atomP(prop("p"))),
                         \mod(iter(atomG(agame("a"))), atomP(prop("q")))) == true;
// <(a* && b)^x>p <= <a*>q
test bool fpLessThanOrEqualTo_test_2() =
       fpLessThanOrEqualTo(\mod(dIter(dChoice(iter(atomG(agame("a"))), atomG(agame("b")))
           ↪ ), atomP(prop("p"))),
                         \mod(iter(atomG(agame("a"))), atomP(prop("q")))) == true;
// <(a* && b)^x><a*>p <= <a*>q
test bool fpLessThanOrEqualTo_test_3() =
       fpLessThanOrEqualTo(\mod(dIter(dChoice(iter(atomG(agame("a"))), atomG(agame("b")))
           ↪ ),
                              \mod(iter(atomG(agame("a"))), atomP(prop("p")))),
                         \mod(iter(atomG(agame("a"))), atomP(prop("q")))) == true;
// <a*><(a* && b)^x>p <= <(a* && b)^x>q
test bool fpLessThanOrEqualTo_test_4() =
       fpLessThanOrEqualTo(\mod(iter(atomG(agame("a"))),
                              \mod(dIter(dChoice(iter(atomG(agame("a"))), atomG(agame("b")
                                  ↪ ))), atomP(prop("p")))),
                         \mod(dIter(dChoice(iter(atomG(agame("a"))), atomG(agame("b")))),
                             ↪  atomP(prop("q")))) == false;
// <a*>p !<= <(a* && b)^x><a*>q
test bool fpLessThanOrEqualTo_test_5() =
       fpLessThanOrEqualTo(\mod(iter(atomG(agame("a"))), atomP(prop("p"))),
                         \mod(dIter(dChoice(iter(atomG(agame("a"))), atomG(agame("b")))),
                              \mod(iter(atomG(agame("a"))), atomP(prop("q"))))) == false;
// <a*>p !<= <a^x>p
test bool fpLessThanOrEqualTo_test_6() =
       fpLessThanOrEqualTo(\mod(iter(atomG(agame("a"))), atomP(prop("p"))),
                         \mod(dIter(atomG(agame("a"))), atomP(prop("p")))) == false;
// <a*>p !<= <a^x>p
test bool fpLessThanOrEqualTo_test_7() =
       fpLessThanOrEqualTo(\mod(iter(atomG(agame("a"))), atomP(prop("p"))),
                         \mod(dIter(atomG(agame("a"))), atomP(prop("p")))) == false;
// <a*^x>p <= <a*>p
test bool fpLessThanOrEqualTo_test_8() =
       fpLessThanOrEqualTo(\mod(dIter(iter(atomG(agame("a")))), atomP(prop("p"))),
                         \mod(iter(atomG(agame("a"))), atomP(prop("p")))) == true;
// p <= q should throw an exception
test bool fpLessThanOrEqualTo_test_9() {
       try fpLessThanOrEqualTo(atomP(prop("p")), atomP(prop("q")));
       catch IllegalArgument("Error: cannot apply fixpoint ordering on non-fixpoint
           ↪ formulae!"): return true;
       return false;
}
// <a>p <= <b>q should throw an exception
test bool fpLessThanOrEqualTo_test_10() {
       try fpLessThanOrEqualTo(\mod(atomG(agame("a")), atomP(prop("p"))), \mod(atomG(
           ↪ agame("b")), atomP(prop("q"))));
       catch IllegalArgument("Error: cannot apply fixpoint ordering on non-fixpoint
           ↪ formulae!"): return true;
       return false;
}
```

C.4.2 ProofSearch_test.rsc

```
module unitTests::ProofSearch_test

import CloG_Base::GLASTs;
import ATP::ATP_Base;
import ATP::ProofSearch;


// Try Closure Discharge

// seq: [ p^[] ]
// cloSeqs: empty
test bool tryDisClo_test_1() {
       CloGSequent input = [term(atomP(prop("p")), [], false)];
       CloSeqs cloSeqs = ();
       MaybeProof output = noProof();

       return tryDisClo(input, cloSeqs) == output;
}

// seq: [ p^[] ]
// cloSeqs: ( x: <[ <a^x>p^[] ], 0> )
test bool tryDisClo_test_2() {
       CloGSequent input = [term(atomP(prop("p")), [], false)];
       CloSeqs cloSeqs = (
              name("x"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
                  ↪ false)], 0>
       );
       MaybeProof output = noProof();
```

```
        return tryDisClo(input, closeqs) == output;
}

// seq: [ <a^x>p^x ]
// cloSeqs: empty
test bool tryDisClo_test_3() {
        CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [name
            ↪ ("x")], false)];
        CloSeqs cloSeqs = ();
        MaybeProof output = noProof();

        return tryDisClo(input, cloSeqs) == output;
}

// seq: [ <a^x>p^x ]
// cloSeqs: ( x: <[ <a^x>p^[] ], 0> )
test bool tryDisClo_test_4() {
        CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [name
            ↪ ("x")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
                    ↪ false)], 0>
        );
        MaybeProof output = proof(disClo(input, name("x")));

        return tryDisClo(input, cloSeqs) == output;
}

// seq: [ <a^x>p^[] ]
// cloSeqs: ( x: <[ <a^x>p^[] ], 0> )
test bool tryDisClo_test_5() {
        CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
            ↪ false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
                    ↪ false)], 0>
        );
        MaybeProof output = noProof();

        return tryDisClo(input, cloSeqs) == output;
}

// seq: [ <a^x>p^y ]
// cloSeqs: ( x: <[ <a^x>p^[] ], 0> )
test bool tryDisClo_test_6() {
        CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [name
            ↪ ("y")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
                    ↪ false)], 0>
        );
        MaybeProof output = noProof();

        return tryDisClo(input, cloSeqs) == output;
```

```
}

// seq: [ <b^x>p^x ]
// cloSeqs: ( x: <[ <a^x>p^[] ], 0> )
test bool tryDisClo_test_7() {
        CloGSequent input = [term(\mod(dIter(atomG(agame("b"))), atomP(prop("p"))), [name
            ↪ ("x")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
                    ↪ false)], 0>
        );
        MaybeProof output = noProof();

        return tryDisClo(input, cloSeqs) == output;
}

// seq: [ <(a||b)^x>p^x ]
// cloSeqs: ( x: <[ <a^x>p^[] ], 0> )
test bool tryDisClo_test_8() {
        CloGSequent input = [term(\mod(dIter(choice(atomG(agame("a")), atomG(agame("b"))))
            ↪ , atomP(prop("p"))), [name("x")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
                    ↪ false)], 0>
        );
        MaybeProof output = noProof();

        return tryDisClo(input, cloSeqs) == output;
}

// seq: [ <(a)^x>p^x ]
// cloSeqs: ( x: <[ <(a||b)^x>p^[] ], 0> )
test bool tryDisClo_test_9() {
        CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [name
            ↪ ("x")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(choice(atomG(agame("a")), atomG(agame("b")))),
                    ↪  atomP(prop("p"))), [], false)], 0>
        );
        MaybeProof output = noProof();

        return tryDisClo(input, cloSeqs) == output;
}


// seq: [ <a^x>p^x ]
// cloSeqs: ( x: <[ q^[] <a^x>p^[] ], 1> )
test bool tryDisClo_test_10() {
        CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [name
            ↪ ("x")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(atomP(prop("q")), [], false), term(\mod(dIter(atomG(agame
                    ↪ ("a"))), atomP(prop("p"))), [], false)], 1>
        );
        MaybeProof output = noProof();

        return tryDisClo(input, cloSeqs) == output;
}

// seq: [ q^[] <a^x>p^x ]
// cloSeqs: ( x: <[ q^[] <a^x>p^[] ], 1> )
test bool tryDisClo_test_11() {
        CloGSequent input = [term(atomP(prop("q")), [], false), term(\mod(dIter(atomG(
            ↪ agame("a"))), atomP(prop("p"))), [name("x")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(atomP(prop("q")), [], false), term(\mod(dIter(atomG(agame
                    ↪ ("a"))), atomP(prop("p"))), [], false)], 1>
        );
        MaybeProof output = proof(disClo(input, name("x")));

        return tryDisClo(input, cloSeqs) == output;
}

// seq: [ q^[] <a^x>p^x ]
// cloSeqs: ( x: <[ <a^x>p^[] q^[] ], 0> )
test bool tryDisClo_test_12() {
        CloGSequent input = [term(atomP(prop("q")), [], false), term(\mod(dIter(atomG(
            ↪ agame("a"))), atomP(prop("p"))), [name("x")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
                    ↪ false), term(atomP(prop("q")), [], false)], 0>
        );
        MaybeProof output = proof(disClo(input, name("x")));

        return tryDisClo(input, cloSeqs) == output;
}

// seq: [ q^[] <a^x>p^x ]
// cloSeqs: ( x: <[ <a^x>p^[] ], 0> )
test bool tryDisClo_test_13() {
        CloGSequent input = [term(atomP(prop("q")), [], false), term(\mod(dIter(atomG(
            ↪ agame("a"))), atomP(prop("p"))), [name("x")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
                    ↪ false)], 0>
        );
        CloGSequent seq1 = [term(atomP(prop("q")), [], true), term(\mod(dIter(atomG(agame
            ↪ ("a"))), atomP(prop("p"))), [name("x")], false)];
        CloGSequent seq2 = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [name("
            ↪ x")], false)];

        MaybeProof output = proof(CloGUnaryInf(seq1, weak(), disClo(seq2, name("x"))));

        return tryDisClo(input, cloSeqs) == output;
}

// seq: [ <a^y>p^y ]
// cloSeqs: ( x: <[ <b^x>q^[] ], 0>, y: <[ <a^x>p^[] ], 0> )
```

```
test bool tryDisClo_test_14() {
        CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [name
            ↪ ("y")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(atomG(agame("b"))), atomP(prop("q"))), [],
                    ↪ false)], 0>,
                name("y"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
                    ↪ false)], 0>
        );
        MaybeProof output = proof(disClo(input, name("y")));

        return tryDisClo(input, cloSeqs) == output;
}

// seq: [ <a^y>p^[x, y] ]
// cloSeqs: ( x: <[ <b^x>q^[] ], 0>, y: <[ <a^x>p^x ], 0> )
test bool tryDisClo_test_15() {
        CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [name
            ↪ ("x"), name("y")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(atomG(agame("b"))), atomP(prop("q"))), [],
                    ↪ false)], 0>,
                name("y"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [name("
                    ↪ x")], false)], 0>
        );
        MaybeProof output = proof(disClo(input, name("y")));

        return tryDisClo(input, cloSeqs) == output;
}


// Detect Cycles

// from: [ ]
// to: [ p^[] ]
test bool detectCycles_test_1() {
        CloGSequent input = [term(atomP(prop("p")), [], false)];
        list[CloGSequent] fpSeqs = [];

        return detectCycles(input, fpSeqs) == false;
}

// from: [ [ p^[] ] ]
// to: [ p^[] ]
test bool detectCycles_test_2() {
        CloGSequent input = [term(atomP(prop("p")), [], false)];
        list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [], false)] ];

        return detectCycles(input, fpSeqs) == true;
}

// from: [ [ p^[] ] ]
// to: [ q^[] ]
test bool detectCycles_test_3() {
        CloGSequent input = [term(atomP(prop("q")), [], false)];
        list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [], false)] ];

        return detectCycles(input, fpSeqs) == false;
}

// from: [ [ p^[] ] ]
// to: [ p^[] q^[] ]
test bool detectCycles_test_4() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [],
            ↪  false)];
        list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [], false)] ];

        return detectCycles(input, fpSeqs) == false;
}

// from: [ [ p^[] q^[] ] ]
// to: [ p^[] ]
test bool detectCycles_test_5() {
        CloGSequent input = [term(atomP(prop("p")), [], false)];
        list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [], false), term(atomP(prop("
            ↪ q")), [], false)] ];

        return detectCycles(input, fpSeqs) == false;
}

// from: [ [ p^[] q^[] ] ]
// to: [ q^[] p^[] ]
test bool detectCycles_test_6() {
        CloGSequent input = [term(atomP(prop("q")), [], false), term(atomP(prop("p")), [],
            ↪  false)];
        list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [], false), term(atomP(prop("
            ↪ q")), [], false)] ];

        return detectCycles(input, fpSeqs) == true;
}

// from: [ [ p^[] ] ]
// to: [ p^x ]
test bool detectCycles_test_7() {
        CloGSequent input = [term(atomP(prop("p")), [name("x")], false)];
        list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [], false)] ];

        return detectCycles(input, fpSeqs) == false;
}

// from: [ [ p^x ] ]
// to: [ p^[] ]
test bool detectCycles_test_8() {
        CloGSequent input = [term(atomP(prop("p")), [], false)];
        list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [name("x")], false)] ];

        return detectCycles(input, fpSeqs) == false;
}
```

```
// from: [ [ p^y ] ]
// to: [ p^[x y] ]
test bool detectCycles_test_9() {
       CloGSequent input = [term(atomP(prop("p")), [name("x"), name("y")], false)];
       list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [name("y")], false)] ];

       return detectCycles(input, fpSeqs) == false;
}

// from: [ [ p^[x y] ] ]
// to: [ p^y ]
test bool detectCycles_test_10() {
       CloGSequent input = [term(atomP(prop("p")), [ name("y")], false)];
       list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [name("y"), name("x")], false
           ↪ )] ];

       return detectCycles(input, fpSeqs) == false;
}

// from: [ [ p^[y x] ] ]
// to: [ p^[x y] ]
test bool detectCycles_test_11() {
       CloGSequent input = [term(atomP(prop("p")), [ name("x"), name("y")], false)];
       list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [name("y"), name("x")], false
           ↪ )] ];

       return detectCycles(input, fpSeqs) == true;
}

// from: [ [ p^[] p^y ] ]
// to: [ p^x p^y ]
test bool detectCycles_test_12() {
       CloGSequent input = [term(atomP(prop("p")), [name("x")], false), term(atomP(prop("
           ↪ p")), [ name("y") ], false)];
       list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [], false), term(atomP(prop("
           ↪ p")), [name("y")], false)] ];

       return detectCycles(input, fpSeqs) == false;
}

// from: [ [ p^[] p^x ] ]
// to: [ p^x p^y ]
test bool detectCycles_test_13() {
       CloGSequent input = [term(atomP(prop("p")), [name("x")], false), term(atomP(prop("
           ↪ p")), [ name("x") ], false)];
       list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [], false), term(atomP(prop("
           ↪ p")), [name("x")], false)] ];

       return detectCycles(input, fpSeqs) == false;
}

// from: [ [ p^x p^y ] ]
// to: [ p^[] p^[x y] ]
```

```
test bool detectCycles_test_14() {
	CloGSequent input = [term(atomP(prop("p")), [ ], false), term(atomP(prop("p")), [
	    ↪ name("x"), name("y") ], false)];
	list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [ name("x") ], false), term(
	    ↪ atomP(prop("p")), [ name("y") ], false)] ];

	return detectCycles(input, fpSeqs) == false;
}

// from: [ [ p^[] p^x q^[] ]
// to: [ p^x q^[] q^y ]
test bool detectCycles_test_15() {
	CloGSequent input = [term(atomP(prop("p")), [name("x")], false), term(atomP(prop("
	    ↪ q")), [], false), term(atomP(prop("q")), [ name ("y") ], false)];
	list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [], false), term(atomP(prop("
	    ↪ p")), [ name("x") ], false), term(atomP(prop("q")), [], false)] ];

	return detectCycles(input, fpSeqs) == false;
}

// from: [ [ p^[] ] [ q^[] ] ]
// to: [ p^[] ]
test bool detectCycles_test_16() {
	CloGSequent input = [term(atomP(prop("p")), [], false)];
	list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [], false)], [term(atomP(prop
	    ↪ ("q")), [], false)] ];

	return detectCycles(input, fpSeqs) == true;
}

// from: [ [ p^[] ] [ q^[] ] [ r^[] ] ]
// to: [ q^[] ]
test bool detectCycles_test_17() {
	CloGSequent input = [term(atomP(prop("q")), [], false)];
	list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [], false)], [term(atomP(prop
	    ↪ ("q")), [], false)], [term(atomP(prop("r")), [], false)] ];

	return detectCycles(input, fpSeqs) == true;
}

// from: [ [ p^[] ] [ q^[x y] r^[] ] [ s^[] ] ]
// to: [ r^[] q^[y x] ]
test bool detectCycles_test_18() {
	CloGSequent input = [term(atomP(prop("r")), [], false), term(atomP(prop("q")), [
	    ↪ name("y"), name("x")], false)];
	list[CloGSequent] fpSeqs = [ [term(atomP(prop("p")), [], false)], [term(atomP(prop
	    ↪ ("q")), [name("x"), name("y")], false), term(atomP(prop("r")), [], false)],
	    ↪  [term(atomP(prop("s")), [], false)] ];

	return detectCycles(input, fpSeqs) == true;
}


// Proof Search with Weakening/Expanding
// from: [ p^[] ]
// to : [ p^[] ]
test bool proofSearchWeakExp_test_1() {
       CloGSequent from = [term(atomP(prop("p")), [], false)];
       CloGSequent to = [term(atomP(prop("p")), [], false)];
       MaybeProof output = proof(CloGUnaryInf(to, weak(), CloGLeaf()));

       return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[] ]
// to : [ q^[] ]
test bool proofSearchWeakExp_test_2() {
       CloGSequent from = [term(atomP(prop("p")), [], false)];
       CloGSequent to = [term(atomP(prop("q")), [], false)];
       MaybeProof output = noProof();

       return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[] q^[] ]
// to : [ p^[] ]
test bool proofSearchWeakExp_test_3() {
       CloGSequent from = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [],
           ↪ false)];
       CloGSequent seq1 = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [],
           ↪ true)];
       CloGSequent to = [term(atomP(prop("p")), [], false)];
       MaybeProof output = proof(CloGUnaryInf(seq1, weak(), CloGUnaryInf(to, weak(),
           ↪ CloGLeaf())));

       return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[] ]
// to : [ p^[] q^[] ]
test bool proofSearchWeakExp_test_4() {
       CloGSequent from = [term(atomP(prop("p")), [], false)];
       CloGSequent to = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [],
           ↪ false)];
       MaybeProof output = noProof();

       return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[] q^[] r^[] ]
// to : [ q^[] ]
test bool proofSearchWeakExp_test_5() {
       CloGSequent from = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [],
           ↪ false), term(atomP(prop("r")), [], false)];
       CloGSequent seq1 = [term(atomP(prop("p")), [], true), term(atomP(prop("q")), [],
           ↪ false), term(atomP(prop("r")), [], false)];
```

```
        CloGSequent seq2 = [term(atomP(prop("q")), [], false), term(atomP(prop("r")), [],
            ↪  true)];
        CloGSequent to = [term(atomP(prop("q")), [], false)];
        MaybeProof output = proof(CloGUnaryInf(seq1, weak(), CloGUnaryInf(seq2, weak(),
            ↪  CloGUnaryInf(to, weak(), CloGLeaf()))));

        return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[] q^[] r^[] ]
// to : [ p^[] r^[] ]
test bool proofSearchWeakExp_test_6() {
        CloGSequent from = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [],
            ↪  false), term(atomP(prop("r")), [], false)];
        CloGSequent seq1 = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [],
            ↪  true), term(atomP(prop("r")), [], false)];
        CloGSequent to = [term(atomP(prop("p")), [], false), term(atomP(prop("r")), [],
            ↪  false)];
        MaybeProof output = proof(CloGUnaryInf(seq1, weak(), CloGUnaryInf(to, weak(),
            ↪  CloGLeaf())));

        return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[] q^[] r^[] ]
// to : [ r^[] p^[] ]
test bool proofSearchWeakExp_test_7() {
        CloGSequent from = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [],
            ↪  false), term(atomP(prop("r")), [], false)];
        CloGSequent seq1 = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [],
            ↪  true), term(atomP(prop("r")), [], false)];
        CloGSequent seq2 = [term(atomP(prop("p")), [], false), term(atomP(prop("r")), [],
            ↪  false)];
        CloGSequent to = [term(atomP(prop("r")), [], false), term(atomP(prop("p")), [],
            ↪  false)];
        MaybeProof output = proof(CloGUnaryInf(seq1, weak(), CloGUnaryInf(seq2, weak(),
            ↪  CloGLeaf())));

        return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[x] ]
// to : [ p^[] ]
test bool proofSearchWeakExp_test_8() {
        CloGSequent from = [term(atomP(prop("p")), [name("x")], false)];
        CloGSequent seq1 = [term(atomP(prop("p")), [name("x")], true)];
        CloGSequent to = [term(atomP(prop("p")), [], false)];
        MaybeProof output = proof(CloGUnaryInf(seq1, exp(), CloGUnaryInf(to, weak(),
            ↪  CloGLeaf())));

        return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[] ]
// to : [ p^[x] ]
test bool proofSearchWeakExp_test_9() {
       CloGSequent from = [term(atomP(prop("p")), [], false)];
       CloGSequent to = [term(atomP(prop("p")), [name("x")], false)];
       MaybeProof output = noProof();

       return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[x y] ]
// to : [ p^[] ]
test bool proofSearchWeakExp_test_10() {
       CloGSequent from = [term(atomP(prop("p")), [name("x"), name("y")], false)];
       CloGSequent seq1 = [term(atomP(prop("p")), [name("x"), name("y")], true)];
       CloGSequent seq2 = [term(atomP(prop("p")), [name("y")], true)];
       CloGSequent to = [term(atomP(prop("p")), [], false)];
       MaybeProof output = proof(CloGUnaryInf(seq1, exp(), CloGUnaryInf(seq2, exp(),
           ↪ CloGUnaryInf(to, weak(), CloGLeaf()))));

       return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[x y] ]
// to : [ p^[x] ]
test bool proofSearchWeakExp_test_11() {
       CloGSequent from = [term(atomP(prop("p")), [name("x"), name("y")], false)];
       CloGSequent seq1 = [term(atomP(prop("p")), [name("x"), name("y")], true)];
       CloGSequent to = [term(atomP(prop("p")), [name("x")], false)];
       MaybeProof output = proof(CloGUnaryInf(seq1, exp(), CloGUnaryInf(to, weak(),
           ↪ CloGLeaf())));

       return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[x y z] ]
// to : [ p^[y] ]
test bool proofSearchWeakExp_test_12() {
       CloGSequent from = [term(atomP(prop("p")), [name("x"), name("y"), name("z")],
           ↪ false)];
       CloGSequent seq1 = [term(atomP(prop("p")), [name("x"), name("y"), name("z")], true
           ↪ )];
       CloGSequent seq2 = [term(atomP(prop("p")), [name("y"), name("z")], true)];
       CloGSequent to = [term(atomP(prop("p")), [name("y")], false)];
       MaybeProof output = proof(CloGUnaryInf(seq1, exp(), CloGUnaryInf(seq2, exp(),
           ↪ CloGUnaryInf(to, weak(), CloGLeaf()))));

       return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[x y z] ]
// to : [ p^[x z] ]
test bool proofSearchWeakExp_test_13() {
       CloGSequent from = [term(atomP(prop("p")), [name("x"), name("y"), name("z")],
           ↪ false)];
```

```
        CloGSequent seq1 = [term(atomP(prop("p")), [name("x"), name("y"), name("z")], true
            ↪ )];
        CloGSequent to = [term(atomP(prop("p")), [name("x"), name("z")], false)];
        MaybeProof output = proof(CloGUnaryInf(seq1, exp(), CloGUnaryInf(to, weak(),
            ↪ CloGLeaf())));

        return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[x y z] ]
// to : [ p^[z x] ]
test bool proofSearchWeakExp_test_14() {
        CloGSequent from = [term(atomP(prop("p")), [name("x"), name("y"), name("z")],
            ↪ false)];
        CloGSequent seq1 = [term(atomP(prop("p")), [name("x"), name("y"), name("z")], true
            ↪ )];
        CloGSequent seq2 = [term(atomP(prop("p")), [name("x"), name("z")], false)];
        CloGSequent to = [term(atomP(prop("p")), [name("z"), name("x")], false)];
        MaybeProof output = proof(CloGUnaryInf(seq1, exp(), CloGUnaryInf(seq2, weak(),
            ↪ CloGLeaf())));

        return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[x] q^[] ]
// to : [ p^[] ]
test bool proofSearchWeakExp_test_15() {
        CloGSequent from = [term(atomP(prop("p")), [name("x")], false), term(atomP(prop("q
            ↪ ")), [], false)];
        CloGSequent seq1 = [term(atomP(prop("p")), [name("x")], true), term(atomP(prop("q
            ↪ ")), [], false)];
        CloGSequent seq2 = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [],
            ↪ true)];
        CloGSequent to = [term(atomP(prop("p")), [], false)];
        MaybeProof output = proof(CloGUnaryInf(seq1, exp(), CloGUnaryInf(seq2, weak(),
            ↪ CloGUnaryInf(to, weak(), CloGLeaf()))));

        return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[] q^[x] ]
// to : [ p^[] ]
test bool proofSearchWeakExp_test_16() {
        CloGSequent from = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [
            ↪ name("x")], false)];
        CloGSequent seq1 = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [
            ↪ name("x")], true)];
        CloGSequent to = [term(atomP(prop("p")), [], false)];
        MaybeProof output = proof(CloGUnaryInf(seq1, weak(), CloGUnaryInf(to, weak(),
            ↪ CloGLeaf())));

        return proofSearchWeakExp(from, to) == output;
}
```

```
// from: [ p^[] q^[x] ]
// to : [ q^[] ]
test bool proofSearchWeakExp_test_17() {
        CloGSequent from = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [
            ↪ name("x")], false)];
        CloGSequent seq1 = [term(atomP(prop("p")), [], true), term(atomP(prop("q")), [name
            ↪ ("x")], false)];
        CloGSequent seq2 = [term(atomP(prop("q")), [name("x")], true)];
        CloGSequent to = [term(atomP(prop("q")), [], false)];
        MaybeProof output = proof(CloGUnaryInf(seq1, weak(), CloGUnaryInf(seq2, exp(),
            ↪ CloGUnaryInf(to, weak(), CloGLeaf()))));

        return proofSearchWeakExp(from, to) == output;
}

// from: [ p^[] q^[x y z] r^[] ]
// to : [ q^[y] ]
test bool proofSearchWeakExp_test_18() {
        CloGSequent from = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [
            ↪ name("x"), name("y"), name("z")], false), term(atomP(prop("r")), [], false)
            ↪ ];
        CloGSequent seq1 = [term(atomP(prop("p")), [], true), term(atomP(prop("q")), [name
            ↪ ("x"), name("y"), name("z")], false), term(atomP(prop("r")), [], false)];
        CloGSequent seq2 = [term(atomP(prop("q")), [name("x"), name("y"), name("z")], true
            ↪ ), term(atomP(prop("r")), [], false)];
        CloGSequent seq3 = [term(atomP(prop("q")), [name("y"), name("z")], true), term(
            ↪ atomP(prop("r")), [], false)];
        CloGSequent seq4 = [term(atomP(prop("q")), [name("y")], false), term(atomP(prop("r
            ↪ ")), [], true)];
        CloGSequent to = [term(atomP(prop("q")), [name("y")], false)];
        MaybeProof output = proof(CloGUnaryInf(seq1, weak(), CloGUnaryInf(seq2, exp(),
            ↪ CloGUnaryInf(seq3, exp(), CloGUnaryInf(seq4, weak(), CloGUnaryInf(to, weak
            ↪ (), CloGLeaf()))))));

        return proofSearchWeakExp(from, to) == output;
}
```

C.4.3 RuleApplications_test.rsc

```
module unitTests::RuleApplications_test

import CloG_Base::GLASTs;
import ATP::RuleApplications;
import ATP::ATP_Base;


// Apply Ax1

// p^[]
test bool applyAx1_test_1() {
        CloGSequent input = [term(atomP(prop("p")), [], false)];
        return applyAx1(input) == noSeq();
```

```
}
// p^[] ~p^[]
test bool applyAx1_test_2() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(neg(atomP(prop("p")))
            ↪ , [], false)];
        return applyAx1(input) == sequent([]);
}
// ~p^[] p^[]
test bool applyAx1_test_3() {
        CloGSequent input = [term(neg(atomP(prop("p"))), [], false), term(atomP(prop("p"))
            ↪ , [], false)];
        return applyAx1(input) == sequent([]);
}
// p^[] ~q^[]
test bool applyAx1_test_4() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(neg(atomP(prop("q")))
            ↪ , [], false)];
        return applyAx1(input) == noSeq();
}
// p^[] ~p^[] q^[]
test bool applyAx1_test_5() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(neg(atomP(prop("p")))
            ↪ , [], false), term(atomP(prop("q")), [], false)];
        return applyAx1(input) == noSeq();
}
// p^[] ~p^x
test bool applyAx1_test_6() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(neg(atomP(prop("p")))
            ↪ , [name("x")], false)];
        return applyAx1(input) == noSeq();
}
// ~p^[] ~(~p)^[]
test bool applyAx1_test_7() {
        CloGSequent input = [term(neg(atomP(prop("p"))), [], false), term(neg(neg(atomP(
            ↪ prop("p")))), [], false)];
        return applyAx1(input) == noSeq();
}
// (p|~p)^[]
test bool applyAx1_test_8() {
        CloGSequent input = [term(or(atomP(prop("p")), neg(atomP(prop("p")))), [], false)
            ↪ ];
        return applyAx1(input) == noSeq();
}

// p^[] ~p^[] q^[]
test bool tryApplyAx1_test_1() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(neg(atomP(prop("p")))
            ↪ , [], false), term(atomP(prop("q")), [], false)];

        CloGSequent seq1 = [term(atomP(prop("p")), [], false), term(neg(atomP(prop("p"))),
            ↪  [], false), term(atomP(prop("q")), [], true)];
        CloGSequent seq2 = [term(atomP(prop("p")), [], true), term(neg(atomP(prop("p"))),
            ↪ [], true)];
```

```
       MaybeProof output = proof(CloGUnaryInf(seq1, weak(), CloGUnaryInf(seq2, ax1(),
           ↪ CloGLeaf())));

       return tryApplyAx1(input) == output;
}
// q^[] p^[] ~p^[]
test bool tryApplyAx1_test_2() {
       CloGSequent input = [term(atomP(prop("q")), [], false), term(atomP(prop("p")), [],
           ↪  false), term(neg(atomP(prop("p"))), [], false)];

       CloGSequent seq1 = [term(atomP(prop("q")), [], true), term(atomP(prop("p")), [],
           ↪ false), term(neg(atomP(prop("p"))), [], false)];
       CloGSequent seq2 = [term(atomP(prop("p")), [], true), term(neg(atomP(prop("p"))),
           ↪ [], true)];

       MaybeProof output = proof(CloGUnaryInf(seq1, weak(), CloGUnaryInf(seq2, ax1(),
           ↪ CloGLeaf())));

       return tryApplyAx1(input) == output;
}
// p^[] q^[] ~p^[]
test bool tryApplyAx1_test_3() {
       CloGSequent input = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [],
           ↪  false), term(neg(atomP(prop("p"))), [], false)];

       CloGSequent seq1 = [term(atomP(prop("p")), [], false), term(atomP(prop("q")), [],
           ↪ true), term(neg(atomP(prop("p"))), [], false)];
       CloGSequent seq2 = [term(atomP(prop("p")), [], true), term(neg(atomP(prop("p"))),
           ↪ [], true)];

       MaybeProof output = proof(CloGUnaryInf(seq1, weak(), CloGUnaryInf(seq2, ax1(),
           ↪ CloGLeaf())));

       return tryApplyAx1(input) == output;
}
// p^x ~p^[]
test bool tryApplyAx1_test_4() {
       CloGSequent input = [term(atomP(prop("p")), [name("x")], false), term(neg(atomP(
           ↪ prop("p"))), [], false)];

       CloGSequent seq1 = [term(atomP(prop("p")), [name("x")], true), term(neg(atomP(prop
           ↪ ("p"))), [], false)];
       CloGSequent seq2 = [term(atomP(prop("p")), [], true), term(neg(atomP(prop("p"))),
           ↪ [], true)];

       MaybeProof output = proof(CloGUnaryInf(seq1, exp(), CloGUnaryInf(seq2, ax1(),
           ↪ CloGLeaf())));

       return tryApplyAx1(input) == output;
}
// p^[] ~p^x
test bool tryApplyAx1_test_5() {
       CloGSequent input = [term(atomP(prop("p")), [], false), term(neg(atomP(prop("p")))
           ↪ , [name("x")], false)];
```

```
        CloGSequent seq1 = [term(atomP(prop("p")), [], false), term(neg(atomP(prop("p"))),
            ↪  [name("x")], true)];
        CloGSequent seq2 = [term(atomP(prop("p")), [], true), term(neg(atomP(prop("p"))),
            ↪ [], true)];

        MaybeProof output = proof(CloGUnaryInf(seq1, exp(), CloGUnaryInf(seq2, ax1(),
            ↪ CloGLeaf())));

        return tryApplyAx1(input) == output;
}
// q^x ~p^[x y] r^[] p^y s^[y z]
test bool tryApplyAx1_test_6() {
        CloGSequent input = [term(atomP(prop("q")), [name("x")], false), term(neg(atomP(
            ↪ prop("p"))), [name("x"), name("y")], false), term(atomP(prop("r")), [],
            ↪ false), term(atomP(prop("p")), [name("y")], false), term(atomP(prop("s")),
            ↪ [name("y"), name("z")], false)];

        CloGSequent seq1 = [term(atomP(prop("q")), [name("x")], true), term(neg(atomP(prop
            ↪ ("p"))), [name("x"), name("y")], false), term(atomP(prop("r")), [], false),
            ↪  term(atomP(prop("p")), [name("y")], false), term(atomP(prop("s")), [name("
            ↪ y"), name("z")], false)];
        CloGSequent seq2 = [term(neg(atomP(prop("p"))), [name("x"), name("y")], true),
            ↪ term(atomP(prop("r")), [], false), term(atomP(prop("p")), [name("y")],
            ↪ false), term(atomP(prop("s")), [name("y"), name("z")], false)];
        CloGSequent seq3 = [term(neg(atomP(prop("p"))), [name("y")], true), term(atomP(
            ↪ prop("r")), [], false), term(atomP(prop("p")), [name("y")], false), term(
            ↪ atomP(prop("s")), [name("y"), name("z")], false)];
        CloGSequent seq4 = [term(neg(atomP(prop("p"))), [], false), term(atomP(prop("r")),
            ↪  [], true), term(atomP(prop("p")), [name("y")], false), term(atomP(prop("s
            ↪ ")), [name("y"), name("z")], false)];
        CloGSequent seq5 = [term(neg(atomP(prop("p"))), [], false), term(atomP(prop("p")),
            ↪  [name("y")], true), term(atomP(prop("s")), [name("y"), name("z")], false)
            ↪ ];
        CloGSequent seq6 = [term(neg(atomP(prop("p"))), [], false), term(atomP(prop("p")),
            ↪  [], false), term(atomP(prop("s")), [name("y"), name("z")], true)];
        CloGSequent seq7 = [term(neg(atomP(prop("p"))), [], true), term(atomP(prop("p")),
            ↪ [], true)];

        MaybeProof output = proof(
                CloGUnaryInf(seq1, weak(),
                        CloGUnaryInf(seq2, exp(),
                                CloGUnaryInf(seq3, exp(),
                                        CloGUnaryInf(seq4, weak(),
                                                CloGUnaryInf(seq5, exp(),
                                                        CloGUnaryInf(seq6, weak(),
                                                                CloGUnaryInf(seq7, ax1(),
                                                                        CloGLeaf()
                                                                )
                                                        )
                                                )
                                        )
                                )
                        )
```

```
               )
       );

       return tryApplyAx1(input) == output;
}


// Apply Modm

// p^[]
test bool applyModm_test_1() {
       CloGSequent input = [term(atomP(prop("p")), [], false)];
       MaybeSequent output = noSeq();

       return applyModm(input) == output;
}
// <a>p^[]
test bool applyModm_test_2() {
       CloGSequent input = [term(\mod(atomG(agame("a")), atomP(prop("p"))), [], false)];
       MaybeSequent output = noSeq();

       return applyModm(input) == output;
}
// <a>p^x <a>q^y
test bool applyModm_test_3() {
       CloGSequent input = [term(\mod(atomG(agame("a")), atomP(prop("p"))), [name("x")],
           ↪ false), term(\mod(atomG(agame("a")), atomP(prop("q"))), [name("y")], false)
           ↪ ];
       MaybeSequent output = noSeq();

       return applyModm(input) == output;
}
// <a>p^x <a^d>q^y
test bool applyModm_test_4() {
       CloGSequent input = [term(\mod(atomG(agame("a")), atomP(prop("p"))), [name("x")],
           ↪ false), term(\mod(dual(atomG(agame("a"))), atomP(prop("q"))), [name("y")],
           ↪ false)];
       MaybeSequent output = sequent([term(atomP(prop("p")), [name("x")], false), term(
           ↪ atomP(prop("q")), [name("y")], false)]);

       return applyModm(input) == output;
}
// <a^d>p^x <a>q^y
test bool applyModm_test_5() {
       CloGSequent input = [term(\mod(dual(atomG(agame("a"))), atomP(prop("p"))), [name("
           ↪ x")], false), term(\mod(atomG(agame("a")), atomP(prop("q"))), [name("y")],
           ↪ false)];
       MaybeSequent output = sequent([term(atomP(prop("p")), [name("x")], false), term(
           ↪ atomP(prop("q")), [name("y")], false)]);

       return applyModm(input) == output;
}
// <a>p^x <b^d>q^y
test bool applyModm_test_6() {
       CloGSequent input = [term(\mod(atomG(agame("a")), atomP(prop("p"))), [name("x")],
           ↪ false), term(\mod(dual(atomG(agame("b"))), atomP(prop("q"))), [name("y")],
           ↪ false)];
       MaybeSequent output = noSeq();

       return applyModm(input) == output;
}
// <a>p^x <a^d>q^y r^[]
test bool applyModm_test_7() {
       CloGSequent input = [term(\mod(atomG(agame("a")), atomP(prop("p"))), [name("x")],
           ↪ false), term(\mod(dual(atomG(agame("a"))), atomP(prop("q"))), [name("y")],
           ↪ false), term(atomP(prop("r")), [], false)];
       MaybeSequent output = noSeq();

       return applyModm(input) == output;
}
// (<a>p | <a^d>q)^x
test bool applyModm_test_8() {
       CloGSequent input = [term(or(\mod(atomG(agame("a")), atomP(prop("p"))), \mod(dual(
           ↪ atomG(agame("a"))), atomP(prop("q")))), [name("x")], false)];
       MaybeSequent output = noSeq();

       return applyModm(input) == output;
}


// Apply Or

// p^[]
test bool applyOr_test_1() {
       CloGSequent input = [term(atomP(prop("p")), [], false)];
       MaybeSequent output = noSeq();

       return applyOr(input, 0) == output;
}
// (p|q)^[]
test bool applyOr_test_2() {
       CloGSequent input = [term(or(atomP(prop("p")), atomP(prop("q"))), [], false)];
       MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(atomP(prop
           ↪ ("q")), [], false)]);

       return applyOr(input, 0) == output;
}
// (p|~p)^[]
test bool applyOr_test_3() {
       CloGSequent input = [term(or(atomP(prop("p")), neg(atomP(prop("p")))), [], false)
           ↪ ];
       MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(neg(atomP(
           ↪ prop("p"))), [], false)]);

       return applyOr(input, 0) == output;
}
// (p|q)^x
test bool applyOr_test_4() {
       CloGSequent input = [term(or(atomP(prop("p")), atomP(prop("q"))), [name("x")],
           ↪ false)];
       MaybeSequent output = sequent([term(atomP(prop("p")), [name("x")], false), term(
           ↪ atomP(prop("q")), [name("x")], false)]);

       return applyOr(input, 0) == output;
}
// [(p|q)^[] r^[]] (apply to first term)
test bool applyOr_test_5() {
       CloGSequent input = [term(or(atomP(prop("p")), atomP(prop("q"))), [], false), term
           ↪ (atomP(prop("r")), [], false)];
       MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(atomP(prop
           ↪ ("q")), [], false), term(atomP(prop("r")), [], false)]);

       return applyOr(input, 0) == output;
}
// [(p|q)^[] r^[]] (apply to second term)
test bool applyOr_test_6() {
       CloGSequent input = [term(or(atomP(prop("p")), atomP(prop("q"))), [], false), term
           ↪ (atomP(prop("r")), [], false)];
       MaybeSequent output = noSeq();

       return applyOr(input, 1) == output;
}
// [p^[] (q|r)^[]] (apply to second term)
test bool applyOr_test_7() {
       CloGSequent input = [term(atomP(prop("p")), [], false), term(or(atomP(prop("q")),
           ↪ atomP(prop("r"))), [], false)];
       MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(atomP(prop
           ↪ ("q")), [], false), term(atomP(prop("r")), [], false)]);

       return applyOr(input, 1) == output;
}
// [p^[] (q|r)^[] s^[]] (apply to second term)
test bool applyOr_test_8() {
       CloGSequent input = [term(atomP(prop("p")), [], false), term(or(atomP(prop("q")),
           ↪ atomP(prop("r"))), [], false), term(atomP(prop("s")), [], false)];
       MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(atomP(prop
           ↪ ("q")), [], false), term(atomP(prop("r")), [], false), term(atomP(prop("s")
           ↪ ), [], false)]);

       return applyOr(input, 1) == output;
}
// [(p|q)^[] (r|s)^[]] (apply to first term)
test bool applyOr_test_9() {
       CloGSequent input = [term(or(atomP(prop("p")), atomP(prop("q"))), [], false), term
           ↪ (or(atomP(prop("r")), atomP(prop("s"))), [], false)];
       MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(atomP(prop
           ↪ ("q")), [], false), term(or(atomP(prop("r")), atomP(prop("s"))), [], false)
           ↪ ]);

       return applyOr(input, 0) == output;
}
// [(p|q)^[] (r|s)^[]] (apply to second term)
```

```
test bool applyOr_test_10() {
        CloGSequent input = [term(or(atomP(prop("p")), atomP(prop("q"))), [], false), term
            ↪ (or(atomP(prop("r")), atomP(prop("s"))), [], false)];
        MaybeSequent output = sequent([term(or(atomP(prop("p")), atomP(prop("q"))), [],
            ↪ false), term(atomP(prop("r")), [], false), term(atomP(prop("s")), [], false
            ↪ )]);

        return applyOr(input, 1) == output;
}
// [(p|q)^[] (r|s)^[]] (apply to both terms)
test bool applyOr_test_11() {
        CloGSequent input = [term(or(atomP(prop("p")), atomP(prop("q"))), [], false), term
            ↪ (or(atomP(prop("r")), atomP(prop("s"))), [], false)];
        MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(atomP(prop
            ↪ ("q")), [], false), term(atomP(prop("r")), [], false), term(atomP(prop("s")
            ↪ ), [], false)]);

        return applyOr(applyOr(input, 0).seq, 2) == output;
}
// [<a>(p|q)^[]]
test bool applyOr_test_12() {
        CloGSequent input = [term(\mod(atomG(agame("a")), or(atomP(prop("p")), atomP(prop
            ↪ ("q")))), [], false)];
        MaybeSequent output = noSeq();

        return applyOr(input, 0) == output;
}


// Apply And

// p^[]
test bool applyAnd_test_1() {
        CloGSequent input = [term(atomP(prop("p")), [], false)];
        MaybeSequents output = noSeqs();

        return applyAnd(input, 0) == output;
}
// (p&q)^[]
test bool applyAnd_test_2() {
        CloGSequent input = [term(and(atomP(prop("p")), atomP(prop("q"))), [], false)];
        MaybeSequents output = sequents([term(atomP(prop("p")), [], false)], [term(atomP(
            ↪ prop("q")), [], false)]);

        return applyAnd(input, 0) == output;
}
// (p&~p)^[]
test bool applyAnd_test_3() {
        CloGSequent input = [term(and(atomP(prop("p")), neg(atomP(prop("p")))), [], false)
            ↪ ];
        MaybeSequents output = sequents([term(atomP(prop("p")), [], false)], [term(neg(
            ↪ atomP(prop("p"))), [], false)]);

        return applyAnd(input, 0) == output;
```

```
}
// (p&q)^x
test bool applyAnd_test_4() {
       CloGSequent input = [term(and(atomP(prop("p")), atomP(prop("q"))), [name("x")],
           ↪ false)];
       MaybeSequents output = sequents([term(atomP(prop("p")), [name("x")], false)], [
           ↪ term(atomP(prop("q")), [name("x")], false)]);

       return applyAnd(input, 0) == output;
}
// [(p&q)^[] r^[]] (apply to first term)
test bool applyAnd_test_5() {
       CloGSequent input = [term(and(atomP(prop("p")), atomP(prop("q"))), [], false),
           ↪ term(atomP(prop("r")), [], false)];
       MaybeSequents output = sequents([term(atomP(prop("p")), [], false), term(atomP(
           ↪ prop("r")), [], false)], [term(atomP(prop("q")), [], false), term(atomP(
           ↪ prop("r")), [], false)]);

       return applyAnd(input, 0) == output;
}
// [(p&q)^[] r^[]] (apply to second term)
test bool applyAnd_test_6() {
       CloGSequent input = [term(and(atomP(prop("p")), atomP(prop("q"))), [], false),
           ↪ term(atomP(prop("r")), [], false)];
       MaybeSequents output = noSeqs();

       return applyAnd(input, 1) == output;
}
// [p^[] (q&r)^[]] (apply to second term)
test bool applyAnd_test_7() {
       CloGSequent input = [term(atomP(prop("p")), [], false), term(and(atomP(prop("q")),
           ↪  atomP(prop("r"))), [], false)];
       MaybeSequents output = sequents([term(atomP(prop("p")), [], false), term(atomP(
           ↪ prop("q")), [], false)], [term(atomP(prop("p")), [], false), term(atomP(
           ↪ prop("r")), [], false)]);

       return applyAnd(input, 1) == output;
}
// [p^[] (q&r)^[] s^[]] (apply to second term)
test bool applyAnd_test_8() {
       CloGSequent input = [term(atomP(prop("p")), [], false), term(and(atomP(prop("q")),
           ↪  atomP(prop("r"))), [], false), term(atomP(prop("s")), [], false)];
       MaybeSequents output = sequents([term(atomP(prop("p")), [], false), term(atomP(
           ↪ prop("q")), [], false), term(atomP(prop("s")), [], false)], [term(atomP(
           ↪ prop("p")), [], false), term(atomP(prop("r")), [], false), term(atomP(prop
           ↪ ("s")), [], false)]);

       return applyAnd(input, 1) == output;
}
// [(p&q)^[] (r&s)^[]] (apply to first term)
test bool applyAnd_test_9() {
       CloGSequent input = [term(and(atomP(prop("p")), atomP(prop("q"))), [], false),
           ↪ term(and(atomP(prop("r")), atomP(prop("s"))), [], false)];
```

```
        MaybeSequents output = sequents([term(atomP(prop("p")), [], false), term(and(atomP
            ↪ (prop("r")), atomP(prop("s"))), [], false)], [term(atomP(prop("q")), [],
            ↪ false), term(and(atomP(prop("r")), atomP(prop("s"))), [], false)]);

        return applyAnd(input, 0) == output;
}
// [(p&q)^[] (r&s)^[]] (apply to second term)
test bool applyAnd_test_10() {
        CloGSequent input = [term(and(atomP(prop("p")), atomP(prop("q"))), [], false),
            ↪ term(and(atomP(prop("r")), atomP(prop("s"))), [], false)];
        MaybeSequents output = sequents([term(and(atomP(prop("p")), atomP(prop("q"))), [],
            ↪  false), term(atomP(prop("r")), [], false)], [term(and(atomP(prop("p")),
            ↪ atomP(prop("q"))), [], false), term(atomP(prop("s")), [], false)]);

        return applyAnd(input, 1) == output;
}
// [(p&q)^[] (r&s)^[]] (apply to both terms)
test bool applyAnd_test_11() {
        CloGSequent input = [term(and(atomP(prop("p")), atomP(prop("q"))), [], false),
            ↪ term(and(atomP(prop("r")), atomP(prop("s"))), [], false)];
        MaybeSequents output1 = sequents([term(atomP(prop("p")), [], false), term(atomP(
            ↪ prop("r")), [], false)], [term(atomP(prop("p")), [], false), term(atomP(
            ↪ prop("s")), [], false)]);
        MaybeSequents output2 = sequents([term(atomP(prop("q")), [], false), term(atomP(
            ↪ prop("r")), [], false)], [term(atomP(prop("q")), [], false), term(atomP(
            ↪ prop("s")), [], false)]);

        initResult = applyAnd(input, 0);
        return applyAnd(initResult.left, 1) == output1 && applyAnd(initResult.right, 1) ==
            ↪  output2;
}
// [<a>(p&q)^[]]
test bool applyAnd_test_12() {
        CloGSequent input = [term(\mod(atomG(agame("a")), and(atomP(prop("p")), atomP(prop
            ↪ ("q")))), [], false)];
        MaybeSequents output = noSeqs();

        return applyAnd(input, 0) == output;
}


// Apply Choice

// p^[]
test bool applyChoice_test_1() {
        CloGSequent input = [term(atomP(prop("p")), [], false)];
        MaybeSequent output = noSeq();

        return applyChoice(input, 0) == output;
}
// <a>p^[]
test bool applyChoice_test_2() {
        CloGSequent input = [term(\mod(atomG(agame("a")), atomP(prop("p"))), [], false)];
        MaybeSequent output = noSeq();
        return applyChoice(input, 0) == output;
}
// <a||b>p^[]
test bool applyChoice_test_3() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [], false)];
        MaybeSequent output = sequent([term(or(\mod(atomG(agame("a")), atomP(prop("p"))),
            ↪ \mod(atomG(agame("b")), atomP(prop("p")))), [], false)]);

        return applyChoice(input, 0) == output;
}
// <a||b>~p^[]
test bool applyChoice_test_4() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), neg(
            ↪ atomP(prop("p")))), [], false)];
        MaybeSequent output = sequent([term(or(\mod(atomG(agame("a")), neg(atomP(prop("p")
            ↪ ))), \mod(atomG(agame("b")), neg(atomP(prop("p"))))), [], false)]);

        return applyChoice(input, 0) == output;
}
// <a||a^d>p^[]
test bool applyChoice_test_5() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), dual(atomG(agame("a")))),
            ↪  atomP(prop("p"))), [], false)];
        MaybeSequent output = sequent([term(or(\mod(atomG(agame("a")), atomP(prop("p"))),
            ↪ \mod(dual(atomG(agame("a"))), atomP(prop("p")))), [], false)]);

        return applyChoice(input, 0) == output;
}
// <a||b>p^x
test bool applyChoice_test_6() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [name("x")], false)];
        MaybeSequent output = sequent([term(or(\mod(atomG(agame("a")), atomP(prop("p"))),
            ↪ \mod(atomG(agame("b")), atomP(prop("p")))), [name("x")], false)]);

        return applyChoice(input, 0) == output;
}
// [<a||b>p^[] q^[]] (apply to first term)
test bool applyChoice_test_7() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [], false), term(atomP(prop("q")), [], false)];
        MaybeSequent output = sequent([term(or(\mod(atomG(agame("a")), atomP(prop("p"))),
            ↪ \mod(atomG(agame("b")), atomP(prop("p")))), [], false), term(atomP(prop("q
            ↪ ")), [], false)]);

        return applyChoice(input, 0) == output;
}
// [<a||b>p^[] q^[]] (apply to second term)
test bool applyChoice_test_8() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [], false), term(atomP(prop("q")), [], false)];
        MaybeSequent output = noSeq();
        return applyChoice(input, 1) == output;
}
// [p^[] <a||b>q^[]] (apply to second term)
test bool applyChoice_test_9() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(choice(atomG(
            ↪ agame("a")), atomG(agame("b"))), atomP(prop("q"))), [], false)];
        MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(or(\mod(
            ↪ atomG(agame("a")), atomP(prop("q"))), \mod(atomG(agame("b")), atomP(prop("q
            ↪ ")))), [], false)]);

        return applyChoice(input, 1) == output;
}
// [p^[] <a||b>q^[] r^[]] (apply to second term)
test bool applyChoice_test_10() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(choice(atomG(
            ↪ agame("a")), atomG(agame("b"))), atomP(prop("q"))), [], false), term(atomP(
            ↪ prop("r")), [], false)];
        MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(or(\mod(
            ↪ atomG(agame("a")), atomP(prop("q"))), \mod(atomG(agame("b")), atomP(prop("q
            ↪ ")))), [], false), term(atomP(prop("r")), [], false)]);

        return applyChoice(input, 1) == output;
}
// [(<a||b>p^[] <c||d>q^[]] (apply to first term)
test bool applyChoice_test_11() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [], false), term(\mod(choice(atomG(agame("c")), atomG(agame("
            ↪ d"))), atomP(prop("q"))), [], false)];
        MaybeSequent output = sequent([term(or(\mod(atomG(agame("a")), atomP(prop("p"))),
            ↪ \mod(atomG(agame("b")), atomP(prop("p")))), [], false), term(\mod(choice(
            ↪ atomG(agame("c")), atomG(agame("d"))), atomP(prop("q"))), [], false)]);

        return applyChoice(input, 0) == output;
}
// [(<a||b>p^[] <c||d>q^[]] (apply to second term)
test bool applyChoice_test_12() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [], false), term(\mod(choice(atomG(agame("c")), atomG(agame("
            ↪ d"))), atomP(prop("q"))), [], false)];
        MaybeSequent output = sequent([term(\mod(choice(atomG(agame("a")), atomG(agame("b
            ↪ "))), atomP(prop("p"))), [], false), term(or(\mod(atomG(agame("c")), atomP(
            ↪ prop("q"))), \mod(atomG(agame("d")), atomP(prop("q")))), [], false)]);

        return applyChoice(input, 1) == output;
}

// [(<a||b>p^[] <c||d>q^[]] (apply to both terms)
test bool applyChoice_test_13() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [], false), term(\mod(choice(atomG(agame("c")), atomG(agame("
            ↪ d"))), atomP(prop("q"))), [], false)];
        MaybeSequent output = sequent([term(or(\mod(atomG(agame("a")), atomP(prop("p"))),
            ↪ \mod(atomG(agame("b")), atomP(prop("p")))), [], false), term(or(\mod(atomG(
            ↪ agame("c")), atomP(prop("q"))), \mod(atomG(agame("d")), atomP(prop("q")))),
            ↪  [], false)]);

        return applyChoice(applyChoice(input, 0).seq, 1) == output;
}
// <a&&b>p^[]
test bool applyChoice_test_14() {
        CloGSequent input = [term(\mod(dChoice(atomG(agame("a")), atomG(agame("b"))),
            ↪ atomP(prop("p"))), [], false)];
        MaybeSequent output = noSeq();

        return applyChoice(input, 0) == output;
}
// <(a||b)*>p^[]
test bool applyChoice_test_15() {
        CloGSequent input = [term(\mod(iter(choice(atomG(agame("a")), atomG(agame("b")))),
            ↪  atomP(prop("p"))), [], false)];
        MaybeSequent output = noSeq();

        return applyChoice(input, 0) == output;
}


// Apply dChoice

// p^[]
test bool applyDChoice_test_1() {
        CloGSequent input = [term(atomP(prop("p")), [], false)];
        MaybeSequent output = noSeq();

        return applyDChoice(input, 0) == output;
}
// <a>p^[]
test bool applyDChoice_test_2() {
        CloGSequent input = [term(\mod(atomG(agame("a")), atomP(prop("p"))), [], false)];
        MaybeSequent output = noSeq();

        return applyDChoice(input, 0) == output;
}
// <a&&b>p^[]
test bool applyDChoice_test_3() {
        CloGSequent input = [term(\mod(dChoice(atomG(agame("a")), atomG(agame("b"))),
            ↪ atomP(prop("p"))), [], false)];
        MaybeSequent output = sequent([term(and(\mod(atomG(agame("a")), atomP(prop("p"))),
            ↪  \mod(atomG(agame("b")), atomP(prop("p")))), [], false)]);

        return applyDChoice(input, 0) == output;
}
// <a&&b>~p^[]
test bool applyDChoice_test_4() {
        CloGSequent input = [term(\mod(dChoice(atomG(agame("a")), atomG(agame("b"))), neg(
            ↪ atomP(prop("p")))), [], false)];
        MaybeSequent output = sequent([term(and(\mod(atomG(agame("a")), neg(atomP(prop("p
            ↪ ")))), \mod(atomG(agame("b")), neg(atomP(prop("p"))))), [], false)]);
        return applyDChoice(input, 0) == output;
}
// <a&&a^d>p^[]
test bool applyDChoice_test_5() {
        CloGSequent input = [term(\mod(dChoice(atomG(agame("a")), dual(atomG(agame("a"))))
            ↪ , atomP(prop("p"))), [], false)];
        MaybeSequent output = sequent([term(and(\mod(atomG(agame("a")), atomP(prop("p"))),
            ↪  \mod(dual(atomG(agame("a"))), atomP(prop("p")))), [], false)]);

        return applyDChoice(input, 0) == output;
}
// <a&&b>p^x
test bool applyDChoice_test_6() {
        CloGSequent input = [term(\mod(dChoice(atomG(agame("a")), atomG(agame("b"))),
            ↪ atomP(prop("p"))), [name("x")], false)];
        MaybeSequent output = sequent([term(and(\mod(atomG(agame("a")), atomP(prop("p"))),
            ↪  \mod(atomG(agame("b")), atomP(prop("p")))), [name("x")], false)]);

        return applyDChoice(input, 0) == output;
}
// [<a&&b>p^[] q^[]] (apply to first term)
test bool applyDChoice_test_7() {
        CloGSequent input = [term(\mod(dChoice(atomG(agame("a")), atomG(agame("b"))),
            ↪ atomP(prop("p"))), [], false), term(atomP(prop("q")), [], false)];
        MaybeSequent output = sequent([term(and(\mod(atomG(agame("a")), atomP(prop("p"))),
            ↪  \mod(atomG(agame("b")), atomP(prop("p")))), [], false), term(atomP(prop("q
            ↪ ")), [], false)]);

        return applyDChoice(input, 0) == output;
}
// [<a&&b>p^[] q^[]] (apply to second term)
test bool applyDChoice_test_8() {
        CloGSequent input = [term(\mod(dChoice(atomG(agame("a")), atomG(agame("b"))),
            ↪ atomP(prop("p"))), [], false), term(atomP(prop("q")), [], false)];
        MaybeSequent output = noSeq();

        return applyDChoice(input, 1) == output;
}
// [p^[] <a&&b>q^[]] (apply to second term)
test bool applyDChoice_test_9() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(dChoice(atomG(
            ↪ agame("a")), atomG(agame("b"))), atomP(prop("q"))), [], false)];
        MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(and(\mod(
            ↪ atomG(agame("a")), atomP(prop("q"))), \mod(atomG(agame("b")), atomP(prop("q
            ↪ ")))), [], false)]);

        return applyDChoice(input, 1) == output;
}
// [p^[] <a&&b>q^[] r^[]] (apply to second term)
test bool applyDChoice_test_10() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(dChoice(atomG(
            ↪ agame("a")), atomG(agame("b"))), atomP(prop("q"))), [], false), term(atomP(
            ↪ prop("r")), [], false)];
```

```
       MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(and(\mod(
           ↪ atomG(agame("a")), atomP(prop("q"))), \mod(atomG(agame("b")), atomP(prop("q
           ↪ ")))), [], false), term(atomP(prop("r")), [], false)]);

       return applyDChoice(input, 1) == output;
}
// [(<a&&b>p^[] <c&&d>q^[]] (apply to first term)
test bool applyDChoice_test_11() {
       CloGSequent input = [term(\mod(dChoice(atomG(agame("a")), atomG(agame("b"))),
           ↪ atomP(prop("p"))), [], false), term(\mod(dChoice(atomG(agame("c")), atomG(
           ↪ agame("d"))), atomP(prop("q"))), [], false)];
       MaybeSequent output = sequent([term(and(\mod(atomG(agame("a")), atomP(prop("p"))),
           ↪  \mod(atomG(agame("b")), atomP(prop("p")))), [], false), term(\mod(dChoice(
           ↪ atomG(agame("c")), atomG(agame("d"))), atomP(prop("q"))), [], false)]);

       return applyDChoice(input, 0) == output;
}
// [(<a&&b>p^[] <c&&d>q^[]] (apply to second term)
test bool applyDChoice_test_12() {
       CloGSequent input = [term(\mod(dChoice(atomG(agame("a")), atomG(agame("b"))),
           ↪ atomP(prop("p"))), [], false), term(\mod(dChoice(atomG(agame("c")), atomG(
           ↪ agame("d"))), atomP(prop("q"))), [], false)];
       MaybeSequent output = sequent([term(\mod(dChoice(atomG(agame("a")), atomG(agame("b
           ↪ "))), atomP(prop("p"))), [], false), term(and(\mod(atomG(agame("c")), atomP
           ↪ (prop("q"))), \mod(atomG(agame("d")), atomP(prop("q")))), [], false)]);

       return applyDChoice(input, 1) == output;
}
// [(<a&&b>p^[] <c&&d>q^[]] (apply to both terms)
test bool applyDChoice_test_13() {
       CloGSequent input = [term(\mod(dChoice(atomG(agame("a")), atomG(agame("b"))),
           ↪ atomP(prop("p"))), [], false), term(\mod(dChoice(atomG(agame("c")), atomG(
           ↪ agame("d"))), atomP(prop("q"))), [], false)];
       MaybeSequent output = sequent([term(and(\mod(atomG(agame("a")), atomP(prop("p"))),
           ↪  \mod(atomG(agame("b")), atomP(prop("p")))), [], false), term(and(\mod(
           ↪ atomG(agame("c")), atomP(prop("q"))), \mod(atomG(agame("d")), atomP(prop("q
           ↪ ")))), [], false)]);

       return applyDChoice(applyDChoice(input, 0).seq, 1) == output;
}
// <a||b>p^[]
test bool applyDChoice_test_14() {
       CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
           ↪ (prop("p"))), [], false)];
       MaybeSequent output = noSeq();

       return applyDChoice(input, 0) == output;
}
// <(a&&b)*>p^[]
test bool applyDChoice_test_15() {
       CloGSequent input = [term(\mod(iter(dChoice(atomG(agame("a")), atomG(agame("b"))))
           ↪ , atomP(prop("p"))), [], false)];
       MaybeSequent output = noSeq();
```

```
        return applyDChoice(input, 0) == output;
}


// Apply concat

// p^[]
test bool applyConcat_test_1() {
        CloGSequent input = [term(atomP(prop("p")), [], false)];
        MaybeSequent output = noSeq();

        return applyConcat(input, 0) == output;
}
// <a>p^[]
test bool applyConcat_test_2() {
        CloGSequent input = [term(\mod(atomG(agame("a")), atomP(prop("p"))), [], false)];
        MaybeSequent output = noSeq();

        return applyConcat(input, 0) == output;
}
// <a;b>p^[]
test bool applyConcat_test_3() {
        CloGSequent input = [term(\mod(concat(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [], false)];
        MaybeSequent output = sequent([term(\mod(atomG(agame("a")), \mod(atomG(agame("b"))
            ↪ , atomP(prop("p")))), [], false)]);

        return applyConcat(input, 0) == output;
}
// <a;b>~p^[]
test bool applyConcat_test_4() {
        CloGSequent input = [term(\mod(concat(atomG(agame("a")), atomG(agame("b"))), neg(
            ↪ atomP(prop("p")))), [], false)];
        MaybeSequent output = sequent([term(\mod(atomG(agame("a")), \mod(atomG(agame("b"))
            ↪ , neg(atomP(prop("p"))))), [], false)]);

        return applyConcat(input, 0) == output;
}
// <a;a^d>p^[]
test bool applyConcat_test_5() {
        CloGSequent input = [term(\mod(concat(atomG(agame("a")), dual(atomG(agame("a")))),
            ↪  atomP(prop("p"))), [], false)];
        MaybeSequent output = sequent([term(\mod(atomG(agame("a")), \mod(dual(atomG(agame
            ↪ ("a"))), atomP(prop("p")))), [], false)]);

        return applyConcat(input, 0) == output;
}
// <a;b>p^x
test bool applyConcat_test_6() {
        CloGSequent input = [term(\mod(concat(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [name("x")], false)];
        MaybeSequent output = sequent([term(\mod(atomG(agame("a")), \mod(atomG(agame("b"))
            ↪ , atomP(prop("p")))), [name("x")], false)]);
```

```
       return applyConcat(input, 0) == output;
}
// [<a;b>p^[] q^[]] (apply to first term)
test bool applyConcat_test_7() {
       CloGSequent input = [term(\mod(concat(atomG(agame("a")), atomG(agame("b"))), atomP
           ↪ (prop("p"))), [], false), term(atomP(prop("q")), [], false)];
       MaybeSequent output = sequent([term(\mod(atomG(agame("a")), \mod(atomG(agame("b"))
           ↪ , atomP(prop("p")))), [], false), term(atomP(prop("q")), [], false)]);

       return applyConcat(input, 0) == output;
}
// [<a;b>p^[] q^[]] (apply to second term)
test bool applyConcat_test_8() {
       CloGSequent input = [term(\mod(concat(atomG(agame("a")), atomG(agame("b"))), atomP
           ↪ (prop("p"))), [], false), term(atomP(prop("q")), [], false)];
       MaybeSequent output = noSeq();

       return applyConcat(input, 1) == output;
}
// [p^[] <a;b>q^[]] (apply to second term)
test bool applyConcat_test_9() {
       CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(concat(atomG(
           ↪ agame("a")), atomG(agame("b"))), atomP(prop("q"))), [], false)];
       MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(\mod(atomG(
           ↪ agame("a")), \mod(atomG(agame("b")), atomP(prop("q")))), [], false)]);

       return applyConcat(input, 1) == output;
}
// [p^[] <a;b>q^[] r^[]] (apply to second term)
test bool applyConcat_test_10() {
       CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(concat(atomG(
           ↪ agame("a")), atomG(agame("b"))), atomP(prop("q"))), [], false), term(atomP(
           ↪ prop("r")), [], false)];
       MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(\mod(atomG(
           ↪ agame("a")), \mod(atomG(agame("b")), atomP(prop("q")))), [], false), term(
           ↪ atomP(prop("r")), [], false)]);

       return applyConcat(input, 1) == output;
}
// [<a;b>p^[] <c;d>q^[]] (apply to first term)
test bool applyConcat_test_11() {
       CloGSequent input = [term(\mod(concat(atomG(agame("a")), atomG(agame("b"))), atomP
           ↪ (prop("p"))), [], false), term(\mod(concat(atomG(agame("c")), atomG(agame("
           ↪ d"))), atomP(prop("q"))), [], false)];
       MaybeSequent output = sequent([term(\mod(atomG(agame("a")), \mod(atomG(agame("b"))
           ↪ , atomP(prop("p")))), [], false), term(\mod(concat(atomG(agame("c")), atomG
           ↪ (agame("d"))), atomP(prop("q"))), [], false)]);

       return applyConcat(input, 0) == output;
}
// [<a;b>p^[] <c;d>q^[]] (apply to second term)
test bool applyConcat_test_12() {
       CloGSequent input = [term(\mod(concat(atomG(agame("a")), atomG(agame("b"))), atomP
           ↪ (prop("p"))), [], false), term(\mod(concat(atomG(agame("c")), atomG(agame("
           ↪ d"))), atomP(prop("q"))), [], false)];
       MaybeSequent output = sequent([term(\mod(concat(atomG(agame("a")), atomG(agame("b
           ↪ "))), atomP(prop("p"))), [], false), term(\mod(atomG(agame("c")), \mod(
           ↪ atomG(agame("d")), atomP(prop("q")))), [], false)]);

       return applyConcat(input, 1) == output;
}
// [<a;b>p^[] <c;d>q^[]] (apply to both terms)
test bool applyConcat_test_13() {
       CloGSequent input = [term(\mod(concat(atomG(agame("a")), atomG(agame("b"))), atomP
           ↪ (prop("p"))), [], false), term(\mod(concat(atomG(agame("c")), atomG(agame("
           ↪ d"))), atomP(prop("q"))), [], false)];
       MaybeSequent output = sequent([term(\mod(atomG(agame("a")), \mod(atomG(agame("b"))
           ↪ , atomP(prop("p")))), [], false), term(\mod(atomG(agame("c")), \mod(atomG(
           ↪ agame("d")), atomP(prop("q")))), [], false)]);

       return applyConcat(applyConcat(input, 0).seq, 1) == output;
}
// <a||b>p^[]
test bool applyConcat_test_14() {
       CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
           ↪ (prop("p"))), [], false)];
       MaybeSequent output = noSeq();

       return applyConcat(input, 0) == output;
}
// <(a;b)*>p^[]
test bool applyConcat_test_15() {
       CloGSequent input = [term(\mod(iter(concat(atomG(agame("a")), atomG(agame("b")))),
           ↪  atomP(prop("p"))), [], false)];
       MaybeSequent output = noSeq();

       return applyConcat(input, 0) == output;
}



// Apply test

// p^[]
test bool applyTest_test_1() {
       CloGSequent input = [term(atomP(prop("p")), [], false)];
       MaybeSequent output = noSeq();

       return applyTest(input, 0) == output;
}
// <a>p^[]
test bool applyTest_test_2() {
       CloGSequent input = [term(\mod(atomG(agame("a")), atomP(prop("p"))), [], false)];
       MaybeSequent output = noSeq();

       return applyTest(input, 0) == output;
}
// <p?>q^[]
```

```
test bool applyTest_test_3() {
        CloGSequent input = [term(\mod(\test(atomP(prop("p"))), atomP(prop("q"))), [],
            ↪ false)];
        MaybeSequent output = sequent([term(and(atomP(prop("p")), atomP(prop("q"))), [],
            ↪ false)]);

        return applyTest(input, 0) == output;
}
// <p?>~q^[]
test bool applyTest_test_4() {
        CloGSequent input = [term(\mod(\test(atomP(prop("p"))), neg(atomP(prop("q")))),
            ↪ [], false)];
        MaybeSequent output = sequent([term(and(atomP(prop("p")), neg(atomP(prop("q")))),
            ↪ [], false)]);

        return applyTest(input, 0) == output;
}
// <~p?>q^[]
test bool applyTest_test_5() {
        CloGSequent input = [term(\mod(\test(neg(atomP(prop("p")))), atomP(prop("q"))),
            ↪ [], false)];
        MaybeSequent output = sequent([term(and(neg(atomP(prop("p"))), atomP(prop("q"))),
            ↪ [], false)]);

        return applyTest(input, 0) == output;
}
// <p?>q^x
test bool applyTest_test_6() {
        CloGSequent input = [term(\mod(\test(atomP(prop("p"))), atomP(prop("q"))), [name("
            ↪ x")], false)];
        MaybeSequent output = sequent([term(and(atomP(prop("p")), atomP(prop("q"))), [name
            ↪ ("x")], false)]);

        return applyTest(input, 0) == output;
}
// [<p?>q^[] r^[]] (apply to first term)
test bool applyTest_test_7() {
        CloGSequent input = [term(\mod(\test(atomP(prop("p"))), atomP(prop("q"))), [],
            ↪ false), term(atomP(prop("r")), [], false)];
        MaybeSequent output = sequent([term(and(atomP(prop("p")), atomP(prop("q"))), [],
            ↪ false), term(atomP(prop("r")), [], false)]);

        return applyTest(input, 0) == output;
}
// [<p?>q^[] r^[]] (apply to second term)
test bool applyTest_test_8() {
        CloGSequent input = [term(\mod(\test(atomP(prop("p"))), atomP(prop("q"))), [],
            ↪ false), term(atomP(prop("r")), [], false)];
        MaybeSequent output = noSeq();

        return applyTest(input, 1) == output;
}
// [p^[] <q?>r^[]] (apply to second term)
test bool applyTest_test_9() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(\test(atomP(prop
            ↪ ("q"))), atomP(prop("r"))), [], false)];
        MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(and(atomP(
            ↪ prop("q")), atomP(prop("r"))), [], false)]);

        return applyTest(input, 1) == output;
}
// [p^[] <q?>r^[] s^[]] (apply to second term)
test bool applyTest_test_10() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(\test(atomP(prop
            ↪ ("q"))), atomP(prop("r"))), [], false), term(atomP(prop("s")), [], false)];
        MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(and(atomP(
            ↪ prop("q")), atomP(prop("r"))), [], false), term(atomP(prop("s")), [], false
            ↪ )]);

        return applyTest(input, 1) == output;
}
// [<p?>q^[] <r?>s^[]] (apply to first term)
test bool applyTest_test_11() {
        CloGSequent input = [term(\mod(\test(atomP(prop("p"))), atomP(prop("q"))), [],
            ↪ false), term(\mod(\test(atomP(prop("r"))), atomP(prop("s"))), [], false)];
        MaybeSequent output = sequent([term(and(atomP(prop("p")), atomP(prop("q"))), [],
            ↪ false), term(\mod(\test(atomP(prop("r"))), atomP(prop("s"))), [], false)]);

        return applyTest(input, 0) == output;
}
// [<p?>q^[] <r?>s^[]] (apply to second term)
test bool applyTest_test_12() {
        CloGSequent input = [term(\mod(\test(atomP(prop("p"))), atomP(prop("q"))), [],
            ↪ false), term(\mod(\test(atomP(prop("r"))), atomP(prop("s"))), [], false)];
        MaybeSequent output = sequent([term(\mod(\test(atomP(prop("p"))), atomP(prop("q"))
            ↪ ), [], false), term(and(atomP(prop("r")), atomP(prop("s"))), [], false)]);

        return applyTest(input, 1) == output;
}
// [<p?>q^[] <r?>s^[]] (apply to both terms)
test bool applyTest_test_13() {
        CloGSequent input = [term(\mod(\test(atomP(prop("p"))), atomP(prop("q"))), [],
            ↪ false), term(\mod(\test(atomP(prop("r"))), atomP(prop("s"))), [], false)];
        MaybeSequent output = sequent([term(and(atomP(prop("p")), atomP(prop("q"))), [],
            ↪ false), term(and(atomP(prop("r")), atomP(prop("s"))), [], false)]);

        return applyTest(applyTest(input, 0).seq, 1) == output;
}
// <a||b>p^[]
test bool applyTest_test_14() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [], false)];
        MaybeSequent output = noSeq();

        return applyTest(input, 0) == output;
}
// <(p?)*>q^[]
test bool applyTest_test_15() {
        CloGSequent input = [term(\mod(iter(\test(atomP(prop("p")))), atomP(prop("q"))),
            ↪ [], false)];
        MaybeSequent output = noSeq();

        return applyTest(input, 0) == output;
}


// Apply dTest

// p^[]
test bool applyDTest_test_1() {
        CloGSequent input = [term(atomP(prop("p")), [], false)];
        MaybeSequent output = noSeq();

        return applyDTest(input, 0) == output;
}
// <a>p^[]
test bool applyDTest_test_2() {
        CloGSequent input = [term(\mod(atomG(agame("a")), atomP(prop("p"))), [], false)];
        MaybeSequent output = noSeq();

        return applyDTest(input, 0) == output;
}
// <p!>q^[]
test bool applyDTest_test_3() {
        CloGSequent input = [term(\mod(dTest(atomP(prop("p"))), atomP(prop("q"))), [],
            ↪ false)];
        MaybeSequent output = sequent([term(\or(atomP(prop("p")), atomP(prop("q"))), [],
            ↪ false)]);

        return applyDTest(input, 0) == output;
}
// <p!>~q^[]
test bool applyDTest_test_4() {
        CloGSequent input = [term(\mod(dTest(atomP(prop("p"))), neg(atomP(prop("q")))),
            ↪ [], false)];
        MaybeSequent output = sequent([term(\or(atomP(prop("p")), neg(atomP(prop("q")))),
            ↪ [], false)]);

        return applyDTest(input, 0) == output;
}
// <~p!>q^[]
test bool applyDTest_test_5() {
        CloGSequent input = [term(\mod(dTest(neg(atomP(prop("p")))), atomP(prop("q"))),
            ↪ [], false)];
        MaybeSequent output = sequent([term(or(neg(atomP(prop("p"))), atomP(prop("q"))),
            ↪ [], false)]);

        return applyDTest(input, 0) == output;
}
// <p!>q^x
test bool applyDTest_test_6() {
        CloGSequent input = [term(\mod(dTest(atomP(prop("p"))), atomP(prop("q"))), [name("
            ↪ x")], false)];
        MaybeSequent output = sequent([term(or(atomP(prop("p")), atomP(prop("q"))), [name
            ↪ ("x")], false)]);

        return applyDTest(input, 0) == output;
}
// [<p!>q^[] r^[]] (apply to first term)
test bool applyDTest_test_7() {
        CloGSequent input = [term(\mod(dTest(atomP(prop("p"))), atomP(prop("q"))), [],
            ↪ false), term(atomP(prop("r")), [], false)];
        MaybeSequent output = sequent([term(or(atomP(prop("p")), atomP(prop("q"))), [],
            ↪ false), term(atomP(prop("r")), [], false)]);

        return applyDTest(input, 0) == output;
}
// [<p!>q^[] r^[]] (apply to second term)
test bool applyDTest_test_8() {
        CloGSequent input = [term(\mod(dTest(atomP(prop("p"))), atomP(prop("q"))), [],
            ↪ false), term(atomP(prop("r")), [], false)];
        MaybeSequent output = noSeq();

        return applyDTest(input, 1) == output;
}
// [p^[] <q!>r^[]] (apply to second term)
test bool applyDTest_test_9() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(dTest(atomP(prop
            ↪ ("q"))), atomP(prop("r"))), [], false)];
        MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(\or(atomP(
            ↪ prop("q")), atomP(prop("r"))), [], false)]);

        return applyDTest(input, 1) == output;
}
// [p^[] <q!>r^[] s^[]] (apply to second term)
test bool applyDTest_test_10() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(dTest(atomP(prop
            ↪ ("q"))), atomP(prop("r"))), [], false), term(atomP(prop("s")), [], false)];
        MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(\or(atomP(
            ↪ prop("q")), atomP(prop("r"))), [], false), term(atomP(prop("s")), [], false
            ↪ )]);

        return applyDTest(input, 1) == output;
}
// [<p!>q^[] <r!>s^[]] (apply to first term)
test bool applyDTest_test_11() {
        CloGSequent input = [term(\mod(dTest(atomP(prop("p"))), atomP(prop("q"))), [],
            ↪ false), term(\mod(dTest(atomP(prop("r"))), atomP(prop("s"))), [], false)];
        MaybeSequent output = sequent([term(or(atomP(prop("p")), atomP(prop("q"))), [],
            ↪ false), term(\mod(dTest(atomP(prop("r"))), atomP(prop("s"))), [], false)]);

        return applyDTest(input, 0) == output;
}
// [<p!>q^[] <r!>s^[]] (apply to second term)
test bool applyDTest_test_12() {
        CloGSequent input = [term(\mod(dTest(atomP(prop("p"))), atomP(prop("q"))), [],
            ↪ false), term(\mod(dTest(atomP(prop("r"))), atomP(prop("s"))), [], false)];
        MaybeSequent output = sequent([term(\mod(dTest(atomP(prop("p"))), atomP(prop("q"))
            ↪ ), [], false), term(or(atomP(prop("r")), atomP(prop("s"))), [], false)]);

        return applyDTest(input, 1) == output;
}
// [<p!>q^[] <r!>s^[]] (apply to both terms)
test bool applyDTest_test_13() {
        CloGSequent input = [term(\mod(dTest(atomP(prop("p"))), atomP(prop("q"))), [],
            ↪ false), term(\mod(dTest(atomP(prop("r"))), atomP(prop("s"))), [], false)];
        MaybeSequent output = sequent([term(or(atomP(prop("p")), atomP(prop("q"))), [],
            ↪ false), term(or(atomP(prop("r")), atomP(prop("s"))), [], false)]);

        return applyDTest(applyDTest(input, 0).seq, 1) == output;
}
// <a||b>p^[]
test bool applyDTest_test_14() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [], false)];
        MaybeSequent output = noSeq();

        return applyDTest(input, 0) == output;
}
// <(p!)*>q^[]
test bool applyDTest_test_15() {
        CloGSequent input = [term(\mod(iter(dTest(atomP(prop("p")))), atomP(prop("q"))),
            ↪ [], false)];
        MaybeSequent output = noSeq();

        return applyDTest(input, 0) == output;
}


// Apply iter

// p^[] (empty closure map)
test bool applyIter_test_1() {
        CloGSequent input = [term(atomP(prop("p")), [], false)];
        MaybeSequent output = noSeq();

        return applyIter(input, 0, ()) == output;
}
// <a>p^[] (empty closure map)
test bool applyIter_test_2() {
        CloGSequent input = [term(\mod(atomG(agame("a")), atomP(prop("p"))), [], false)];
        MaybeSequent output = noSeq();

        return applyIter(input, 0, ()) == output;
}
// <a*>p^[] (empty closure map)
test bool applyIter_test_3() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [],
            ↪ false)];
        MaybeSequent output = sequent([term(or(atomP(prop("p")), \mod(atomG(agame("a")), \
            ↪ mod(iter(atomG(agame("a"))), atomP(prop("p"))))), [], false)]);

        return applyIter(input, 0, ()) == output;
}
// <a*>p^[] (closure map contains fp formula for x: <a^x>p)
test bool applyIter_test_4() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [],
            ↪ false)];
        CloSeqs cloSeqs = (name("x"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p
            ↪ "))), [], false)], 0>);
        MaybeSequent output = sequent([term(or(atomP(prop("p")), \mod(atomG(agame("a")), \
            ↪ mod(iter(atomG(agame("a"))), atomP(prop("p"))))), [], false)]);

        return applyIter(input, 0, cloSeqs) == output;
}
// <a*>~p^[] (empty closure map)
test bool applyIter_test_5() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), neg(atomP(prop("p")))),
            ↪ [], false)];
        MaybeSequent output = sequent([term(or(neg(atomP(prop("p"))), \mod(atomG(agame("a
            ↪ ")), \mod(iter(atomG(agame("a"))), neg(atomP(prop("p")))))), [], false)]);

        return applyIter(input, 0, ()) == output;
}
// <(a^d)*>p^[] (empty closure map)
test bool applyIter_test_6() {
        CloGSequent input = [term(\mod(iter(dual(atomG(agame("a")))), atomP(prop("p"))),
            ↪ [], false)];
        MaybeSequent output = sequent([term(or(atomP(prop("p")), \mod(dual(atomG(agame("a
            ↪ "))), \mod(iter(dual(atomG(agame("a")))), atomP(prop("p"))))), [], false)])
            ↪ ;

        return applyIter(input, 0, ()) == output;
}
// <a*>p^x (closure map contains fp formula for x: <a*>p == <a*>p)
test bool applyIter_test_7() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [name("
            ↪ x")], false)];
        CloSeqs cloSeqs = (name("x"): <[term(\mod(iter(atomG(agame("a"))), atomP(prop("p")
            ↪ )), [], false)], 0>);
        MaybeSequent output = sequent([term(or(atomP(prop("p")), \mod(atomG(agame("a")), \
            ↪ mod(iter(atomG(agame("a"))), atomP(prop("p"))))), [name("x")], false)]);

        return applyIter(input, 0, cloSeqs) == output;
}
// <a*>p^x (closure map contains fp formula for x: <a*^x>p < <a*>p)
test bool applyIter_test_8() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [name("
            ↪ x")], false)];
        CloSeqs cloSeqs = (name("x"): <[term(\mod(dIter(iter(atomG(agame("a")))), atomP(
            ↪ prop("p"))), [], false)], 0>);
        MaybeSequent output = sequent([term(or(atomP(prop("p")), \mod(atomG(agame("a")), \
            ↪ mod(iter(atomG(agame("a"))), atomP(prop("p"))))), [name("x")], false)]);
        return applyIter(input, 0, cloSeqs) == output;
}
// <a*>p^x (closure map contains fp formula for x: <a^x>p !<= <a*>p)
test bool applyIter_test_9() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [name("
            ↪ x")], false)];
        CloSeqs cloSeqs = (name("x"): <[term(\mod(dIter(atomG(agame("b"))), atomP(prop("p
            ↪ "))), [], false)], 0>);
        MaybeSequent output = noSeq();

        return applyIter(input, 0, cloSeqs) == output;
}
// <a*>p^y (closure map contains fp formula for x: <a^x>p !<= <a*>p and y: <a*^x>p < <a*>
    ↪ p)
test bool applyIter_test_10() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [name("
            ↪ y")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
                    ↪ false)], 0>,
                name("y"): <[term(\mod(dIter(iter(atomG(agame("a")))), atomP(prop("p"))),
                    ↪ [], false)], 0>
        );
        MaybeSequent output = sequent([term(or(atomP(prop("p")), \mod(atomG(agame("a")), \
            ↪ mod(iter(atomG(agame("a"))), atomP(prop("p"))))), [name("y")], false)]);

        return applyIter(input, 0, cloSeqs) == output;
}
// <a*>p^x (closure map contains fp formula for x: <a*^x>p < <a*>p and y: <a^x>p !<= <a*>
    ↪ p)
test bool applyIter_test_11() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [name("
            ↪ x")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(iter(atomG(agame("a")))), atomP(prop("p"))),
                    ↪ [], false)], 0>,
                name("y"): <[term(\mod(iter(atomG(agame("b"))), atomP(prop("p"))), [],
                    ↪ false)], 0>
        );
        MaybeSequent output = sequent([term(or(atomP(prop("p")), \mod(atomG(agame("a")), \
            ↪ mod(iter(atomG(agame("a"))), atomP(prop("p"))))), [name("x")], false)]);

        return applyIter(input, 0, cloSeqs) == output;
}
// <a*>p^{x, y} (closure map contains fp formula for x: <a*^x>p <= <a*>p and y: <(a*;b)^x
    ↪ >q < <a*>p)
test bool applyIter_test_12() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [name("
            ↪ x"), name("y")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(iter(atomG(agame("a")))), atomP(prop("p"))),
                    ↪ [], false)], 0>,
                name("y"): <[term(\mod(dIter(concat(iter(atomG(agame("a"))), atomG(agame("b
                    ↪ ")))), atomP(prop("q"))), [name("x")], false)], 0>
        );
        MaybeSequent output = sequent([term(or(atomP(prop("p")), \mod(atomG(agame("a")), \
            ↪ mod(iter(atomG(agame("a"))), atomP(prop("p"))))), [name("x"), name("y")],
            ↪ false)]);

        return applyIter(input, 0, cloSeqs) == output;
}
// <a*>p^{x, y} (closure map contains fp formula for x: <a*^x>p < <a*>p and y: <a^x>p !<=
    ↪  <a*>p)
test bool applyIter_test_13() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [name("
            ↪ x"), name("y")], false)];
        CloSeqs cloSeqs = (
                name("x"): <[term(\mod(dIter(iter(atomG(agame("a")))), atomP(prop("p"))),
                    ↪ [], false)], 0>,
                name("y"): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [name("
                    ↪ x")], false)], 0>
        );
        MaybeSequent output = noSeq();

        return applyIter(input, 0, cloSeqs) == output;
}
// [<a*>p^[] q^[]] (apply to first term; empty closure map)
test bool applyIter_test_14() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [],
            ↪ false), term(atomP(prop("q")), [], false)];
        MaybeSequent output = sequent([term(or(atomP(prop("p")), \mod(atomG(agame("a")), \
            ↪ mod(iter(atomG(agame("a"))), atomP(prop("p"))))), [], false), term(atomP(
            ↪ prop("q")), [], false)]);

        return applyIter(input, 0, ()) == output;
}
// [<a*>p^[] q^[]] (apply to second term; empty closure map)
test bool applyIter_test_15() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [],
            ↪ false), term(atomP(prop("q")), [], false)];
        MaybeSequent output = noSeq();

        return applyIter(input, 1, ()) == output;
}
// [p^[] <a*>q^[]] (apply to second term; empty closure map)
test bool applyIter_test_16() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(iter(atomG(agame
            ↪ ("a"))), atomP(prop("q"))), [], false)];
        MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(or(atomP(
            ↪ prop("q")), \mod(atomG(agame("a")), \mod(iter(atomG(agame("a"))), atomP(
            ↪ prop("q"))))), [], false)]);

        return applyIter(input, 1, ()) == output;
}
// [p^[] <a*>q^[] r^[]] (apply to second term; empty closure map)
test bool applyIter_test_17() {
        CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(iter(atomG(agame
            ↪ ("a"))), atomP(prop("q"))), [], false), term(atomP(prop("r")), [], false)];
        MaybeSequent output = sequent([term(atomP(prop("p")), [], false), term(or(atomP(
            ↪ prop("q")), \mod(atomG(agame("a")), \mod(iter(atomG(agame("a"))), atomP(
            ↪ prop("q"))))), [], false), term(atomP(prop("r")), [], false)]);

        return applyIter(input, 1, ()) == output;
}
// [<a*>p^[] <b*>q^[]] (apply to first term; empty closure map)
test bool applyIter_test_18() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [],
            ↪ false), term(\mod(iter(atomG(agame("b"))), atomP(prop("q"))), [], false)];
        MaybeSequent output = sequent([term(or(atomP(prop("p")), \mod(atomG(agame("a")), \
            ↪ mod(iter(atomG(agame("a"))), atomP(prop("p"))))), [], false), term(\mod(
            ↪ iter(atomG(agame("b"))), atomP(prop("q"))), [], false)]);

        return applyIter(input, 0, ()) == output;
}
// [<a*>p^[] <b*>q^[]] (apply to second term; empty closure map)
test bool applyIter_test_19() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [],
            ↪ false), term(\mod(iter(atomG(agame("b"))), atomP(prop("q"))), [], false)];
        MaybeSequent output = sequent([term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))
            ↪ ), [], false), term(or(atomP(prop("q")), \mod(atomG(agame("b")), \mod(iter(
            ↪ atomG(agame("b"))), atomP(prop("q"))))), [], false)]);

        return applyIter(input, 1, ()) == output;
}
// [<a*>p^[] <b*>q^[]] (apply to both terms; empty closure map)
test bool applyIter_test_20() {
        CloGSequent input = [term(\mod(iter(atomG(agame("a"))), atomP(prop("p"))), [],
            ↪ false), term(\mod(iter(atomG(agame("b"))), atomP(prop("q"))), [], false)];
        MaybeSequent output = sequent([term(or(atomP(prop("p")), \mod(atomG(agame("a")), \
            ↪ mod(iter(atomG(agame("a"))), atomP(prop("p"))))), [], false), term(or(atomP
            ↪ (prop("q")), \mod(atomG(agame("b")), \mod(iter(atomG(agame("b"))), atomP(
            ↪ prop("q"))))), [], false)]);

        return applyIter(applyIter(input, 0, ()).seq, 1, ()) == output;
}
// <a||b>p^[]
test bool applyIter_test_21() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [], false)];
        MaybeSequent output = noSeq();

        return applyIter(input, 0, ()) == output;
}
// <(a*)^x>p^[]
test bool applyIter_test_22() {
        CloGSequent input = [term(\mod(dIter(iter(atomG(agame("a")))), atomP(prop("p"))),
            ↪ [], false)];
        MaybeSequent output = noSeq();

        return applyIter(input, 0, ()) == output;
```

```
}


// Apply clo

// p^[] (empty closure map)
test bool applyClo_test_1() {
       CloGSequent input = [term(atomP(prop("p")), [], false)];
       tuple[MaybeSequent, CloGName] output = <noSeq(), name("")>;

       return applyClo(input, 0, ()) == output;
}
// <a>p^[] (empty closure map)
test bool applyClo_test_2() {
       CloGSequent input = [term(\mod(atomG(agame("a")), atomP(prop("p"))), [], false)];
       tuple[MaybeSequent, CloGName] output = <noSeq(), name("")>;

       return applyClo(input, 0, ()) == output;
}
// <a^x>p^[] (empty closure map)
test bool applyClo_test_3() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
           ↪ false)];
       tuple[MaybeSequent, CloGName] output = <sequent([term(and(atomP(prop("p")), \mod(
           ↪ atomG(agame("a")), \mod(dIter(atomG(agame("a"))), atomP(prop("p"))))), [
           ↪ nameS("x",0)], false)]), nameS("x", 0)>;

       return applyClo(input, 0, ()) == output;
}
// <a^x>p^[] (closure map contains fp formula for x_0: <b^x>p)
test bool applyClo_test_10() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
           ↪ false)];
       CloSeqs cloSeqs = (nameS("x", 0): <[term(\mod(dIter(atomG(agame("b"))), atomP(prop
           ↪ ("p"))), [], false)], 0>);
       tuple[MaybeSequent, CloGName] output = <sequent([term(and(atomP(prop("p")), \mod(
           ↪ atomG(agame("a")), \mod(dIter(atomG(agame("a"))), atomP(prop("p"))))), [
           ↪ nameS("x", 1)], false)]), nameS("x", 1)>;

       return applyClo(input, 0, cloSeqs) == output;
}
// <a^x>~p^[] (empty closure map)
test bool applyClo_test_11() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), neg(atomP(prop("p")))),
           ↪ [], false)];
       tuple[MaybeSequent, CloGName] output = <sequent([term(and(neg(atomP(prop("p"))), \
           ↪ mod(atomG(agame("a")), \mod(dIter(atomG(agame("a"))), neg(atomP(prop("p")))
           ↪ ))), [nameS("x", 0)], false)]), nameS("x", 0)>;

       return applyClo(input, 0, ()) == output;
}
// <(a^d)^x>p^[] (empty closure map)
test bool applyClo_test_12() {
       CloGSequent input = [term(\mod(dIter(dual(atomG(agame("a")))), atomP(prop("p"))),
           ↪ [], false)];
       tuple[MaybeSequent, CloGName] output = <sequent([term(and(atomP(prop("p")), \mod(
           ↪ dual(atomG(agame("a"))), \mod(dIter(dual(atomG(agame("a")))), atomP(prop("p
           ↪ "))))), [nameS("x", 0)], false)]), nameS("x", 0)>;

       return applyClo(input, 0, ()) == output;
}
// <a^x>p^x_0 (closure map contains fp formula for x_0: <a^x>p == <a^x>p)
test bool applyClo_test_13() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [nameS
           ↪ ("x", 0)], false)];
       CloSeqs cloSeqs = (nameS("x", 0): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop
           ↪ ("p"))), [], false)], 0>);
       tuple[MaybeSequent, CloGName] output = <sequent([term(and(atomP(prop("p")), \mod(
           ↪ atomG(agame("a")), \mod(dIter(atomG(agame("a"))), atomP(prop("p"))))), [
           ↪ nameS("x", 0), nameS("x", 1)], false)]), nameS("x", 1)>;

       return applyClo(input, 0, cloSeqs) == output;
}
// <a^x>p^x_0 (closure map contains fp formula for x_0: <a^x^x>p < <a^x>p)
test bool applyClo_test_14() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [nameS
           ↪ ("x", 0)], false)];
       CloSeqs cloSeqs = (nameS("x", 0): <[term(\mod(dIter(dIter(atomG(agame("a")))),
           ↪ atomP(prop("p"))), [], false)], 0>);
       tuple[MaybeSequent, CloGName] output = <sequent([term(and(atomP(prop("p")), \mod(
           ↪ atomG(agame("a")), \mod(dIter(atomG(agame("a"))), atomP(prop("p"))))), [
           ↪ nameS("x", 0), nameS("x", 1)], false)]), nameS("x", 1)>;

       return applyClo(input, 0, cloSeqs) == output;
}
// <a^x>p^x_0 (closure map contains fp formula for x_0: <b^x>p !<= <a^x>p)
test bool applyClo_test_15() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [nameS
           ↪ ("x", 0)], false)];
       CloSeqs cloSeqs = (nameS("x", 0): <[term(\mod(dIter(atomG(agame("b"))), atomP(prop
           ↪ ("p"))), [], false)], 0>);
       tuple[MaybeSequent, CloGName] output = <noSeq(), name("")>;

       return applyClo(input, 0, cloSeqs) == output;
}
// <a^x>p^x_1 (closure map contains fp formula for x_0: <b^x>p !<= <a^x>p and x_1: <a^x^x
    ↪ >p < <a^x>p)
test bool applyClo_test_16() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [nameS
           ↪ ("x", 1)], false)];
       CloSeqs cloSeqs = (
               nameS("x", 0): <[term(\mod(dIter(atomG(agame("b"))), atomP(prop("p"))), [],
                   ↪  false)], 0>,
               nameS("x", 1): <[term(\mod(dIter(dIter(atomG(agame("a")))), atomP(prop("p")
                   ↪ )), [], false)], 0>
       );
       tuple[MaybeSequent, CloGName] output = <sequent([term(and(atomP(prop("p")), \mod(
           ↪ atomG(agame("a")), \mod(dIter(atomG(agame("a"))), atomP(prop("p"))))), [
           ↪ nameS("x", 1), nameS("x", 2)], false)]), nameS("x", 2)>;

       return applyClo(input, 0, cloSeqs) == output;
}
// <a^x>p^x_0 (closure map contains fp formula for x_0: <a^x^x>p <= <a^x>p and x_1: <b^x>
    ↪ p !<= <a^x>p)
test bool applyClo_test_17() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [nameS
           ↪ ("x", 0)], false)];
       CloSeqs cloSeqs = (
               nameS("x", 0): <[term(\mod(dIter(dIter(atomG(agame("a")))), atomP(prop("p")
                   ↪ )), [], false)], 0>,
               nameS("x", 1): <[term(\mod(dIter(atomG(agame("b"))), atomP(prop("p"))), [],
                   ↪  false)], 0>
       );
       tuple[MaybeSequent, CloGName] output = <sequent([term(and(atomP(prop("p")), \mod(
           ↪ atomG(agame("a")), \mod(dIter(atomG(agame("a"))), atomP(prop("p"))))), [
           ↪ nameS("x", 0), nameS("x", 2)], false)]), nameS("x", 2)>;

       return applyClo(input, 0, cloSeqs) == output;
}
// <a^x>p^{x_0, x_1} (closure map contains fp formula for x_0: <a^x^x>p < <a^x>p and x_1:
    ↪  <(a^x;b)^x>q < <a^x>p)
test bool applyClo_test_18() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [nameS
           ↪ ("x", 0), nameS("x", 1)], false)];
       CloSeqs cloSeqs = (
               nameS("x", 0): <[term(\mod(dIter(dIter(atomG(agame("a")))), atomP(prop("p")
                   ↪ )), [], false)], 0>,
               nameS("x", 1): <[term(\mod(dIter(concat(dIter(atomG(agame("a"))), atomG(
                   ↪ agame("b")))), atomP(prop("q"))), [nameS("x", 0)], false)], 0>
       );
       tuple[MaybeSequent, CloGName] output = <sequent([term(and(atomP(prop("p")), \mod(
           ↪ atomG(agame("a")), \mod(dIter(atomG(agame("a"))), atomP(prop("p"))))), [
           ↪ nameS("x", 0), nameS("x", 1), nameS("x", 2)], false)]), nameS("x", 2)>;

       return applyClo(input, 0, cloSeqs) == output;
}
// <a^x>p^{x_0, x_1} (closure map contains fp formula for x_0: <a^x^x>p < <a^x>p and x_1:
    ↪  <b^x>p !<= <a^x>p)
test bool applyClo_test_19() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [nameS
           ↪ ("x", 0), nameS("x", 1)], false)];
       CloSeqs cloSeqs = (
               nameS("x", 0): <[term(\mod(dIter(dIter(atomG(agame("a")))), atomP(prop("p")
                   ↪ )), [], false)], 0>,
               nameS("x", 1): <[term(\mod(dIter(atomG(agame("b"))), atomP(prop("p"))), [
                   ↪ nameS("x", 0)], false)], 0>
       );
       tuple[MaybeSequent, CloGName] output = <noSeq(), name("")>;

       return applyClo(input, 0, cloSeqs) == output;
```

```
}
// [<a^x>p^[] q^[]] (apply to first term; empty closure map)
test bool applyClo_test_20() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
           ↪ false), term(atomP(prop("q")), [], false)];
       tuple[MaybeSequent, CloGName] output = <sequent([term(and(atomP(prop("p")), \mod(
           ↪ atomG(agame("a")), \mod(dIter(atomG(agame("a"))), atomP(prop("p"))))), [
           ↪ nameS("x", 0)], false), term(atomP(prop("q")), [], false)]), nameS("x", 0)
           ↪ >;

       return applyClo(input, 0, ()) == output;
}
// [<a^x>p^[] q^[]] (apply to second term; empty closure map)
test bool applyClo_test_21() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
           ↪ false), term(atomP(prop("q")), [], false)];
       tuple[MaybeSequent, CloGName] output = <noSeq(), name("")>;

       return applyClo(input, 1, ()) == output;
}
// [p^[] <a^x>q^[]] (apply to second term; empty closure map)
test bool applyClo_test_22() {
       CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(dIter(atomG(
           ↪ agame("a"))), atomP(prop("q"))), [], false)];
       tuple[MaybeSequent, CloGName] output = <sequent([term(atomP(prop("p")), [], false)
           ↪ , term(and(atomP(prop("q")), \mod(atomG(agame("a")), \mod(dIter(atomG(agame
           ↪ ("a"))), atomP(prop("q"))))), [nameS("x", 0)], false)]), nameS("x", 0)>;

       return applyClo(input, 1, ()) == output;
}
// [p^[] <a^x>q^[] r^[]] (apply to second term; empty closure map)
test bool applyClo_test_23() {
       CloGSequent input = [term(atomP(prop("p")), [], false), term(\mod(dIter(atomG(
           ↪ agame("a"))), atomP(prop("q"))), [], false), term(atomP(prop("r")), [],
           ↪ false)];
       tuple[MaybeSequent, CloGName] output = <sequent([term(atomP(prop("p")), [], false)
           ↪ , term(and(atomP(prop("q")), \mod(atomG(agame("a")), \mod(dIter(atomG(agame
           ↪ ("a"))), atomP(prop("q"))))), [nameS("x", 0)], false), term(atomP(prop("r")
           ↪ ), [], false)]), nameS("x", 0)>;

       return applyClo(input, 1, ()) == output;
}
// [<a^x>p^[] <b^x>q^[]] (apply to first term; empty closure map)
test bool applyClo_test_24() {
       CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
           ↪ false), term(\mod(dIter(atomG(agame("b"))), atomP(prop("q"))), [], false)];
       tuple[MaybeSequent, CloGName] output = <sequent([term(and(atomP(prop("p")), \mod(
           ↪ atomG(agame("a")), \mod(dIter(atomG(agame("a"))), atomP(prop("p"))))), [
           ↪ nameS("x", 0)], false), term(\mod(dIter(atomG(agame("b"))), atomP(prop("q")
           ↪ )), [], false)]), nameS("x", 0)>;

       return applyClo(input, 0, ()) == output;
}
// [<a^x>p^[] <b^x>q^[]] (apply to second term; empty closure map)
```

```
test bool applyClo_test_25() {
        CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
            ↪ false), term(\mod(dIter(atomG(agame("b"))), atomP(prop("q"))), [], false)];
        tuple[MaybeSequent, CloGName] output = <sequent([term(\mod(dIter(atomG(agame("a"))
            ↪ ), atomP(prop("p"))), [], false), term(and(atomP(prop("q")), \mod(atomG(
            ↪ agame("b")), \mod(dIter(atomG(agame("b"))), atomP(prop("q"))))), [nameS("x
            ↪ ", 0)], false)]), nameS("x", 0)>;

        return applyClo(input, 1, ()) == output;
}
// [<a^x>p^[] <b^x>q^[]] (apply to both terms; empty closure map)
test bool applyClo_test_26() {
        CloGSequent input = [term(\mod(dIter(atomG(agame("a"))), atomP(prop("p"))), [],
            ↪ false), term(\mod(dIter(atomG(agame("b"))), atomP(prop("q"))), [], false)];
        tuple[MaybeSequent, CloGName] output1 = <sequent([term(and(atomP(prop("p")), \mod(
            ↪ atomG(agame("a")), \mod(dIter(atomG(agame("a"))), atomP(prop("p"))))), [
            ↪ nameS("x", 0)], false), term(\mod(dIter(atomG(agame("b"))), atomP(prop("q")
            ↪ )), [], false)]), nameS("x", 0)>;
        CloSeqs cloSeqs = (nameS("x", 0): <[term(\mod(dIter(atomG(agame("a"))), atomP(prop
            ↪ ("p"))), [], false)], 0>);
        tuple[MaybeSequent, CloGName] output2 = <sequent([term(and(atomP(prop("p")), \mod(
            ↪ atomG(agame("a")), \mod(dIter(atomG(agame("a"))), atomP(prop("p"))))), [
            ↪ nameS("x", 0)], false), term(and(atomP(prop("q")), \mod(atomG(agame("b")),
            ↪ \mod(dIter(atomG(agame("b"))), atomP(prop("q"))))), [nameS("x", 1)], false)
            ↪ ]), nameS("x", 1)>;

        tuple[MaybeSequent, CloGName] result1 = applyClo(input, 0, ());
        tuple[MaybeSequent, CloGName] result2 = applyClo(result1[0].seq, 1, cloSeqs);

        return result1 == output1 && result2 == output2;
}
// [<a^x^x>p^[]] (apply twice (with an and application in between))
test bool applyClo_test_28() {
        CloGSequent input = [term(\mod(dIter(dIter(atomG(agame("a")))), atomP(prop("p"))),
            ↪  [], false)];
        tuple[MaybeSequent, CloGName] output1 = <sequent([term(and(atomP(prop("p")), \mod(
            ↪ dIter(atomG(agame("a"))), \mod(dIter(dIter(atomG(agame("a")))), atomP(prop
            ↪ ("p"))))), [nameS("x", 0)], false)]), nameS("x", 0)>;
        CloSeqs cloSeqs = (nameS("x", 0): <[term(\mod(dIter(dIter(atomG(agame("a")))),
            ↪ atomP(prop("p"))), [], false)], 0>);
        tuple[MaybeSequent, CloGName] output2 = <sequent([term(and(\mod(dIter(dIter(atomG(
            ↪ agame("a")))), atomP(prop("p"))),\mod(atomG(agame("a")), \mod(dIter(atomG(
            ↪ agame("a"))), \mod(dIter(dIter(atomG(agame("a")))), atomP(prop("p")))))), [
            ↪ nameS("x", 0), nameS("x", 1)],false)]), nameS("x", 1)>;

        tuple[MaybeSequent, CloGName] result1 = applyClo(input, 0, ());
        MaybeSequents result2 = applyAnd(result1[0].seq, 0);
        tuple[MaybeSequent, CloGName] result3 = applyClo(result2.right, 0, cloSeqs);

        return result1 == output1 && result3 == output2;
}
// [<a^x^x>p^[]] (apply thrice (with two and applications in between))
// should not work since <a^x><a^x^x>p^[x_0] is in the cloSeq after the 2nd clo
    ↪ application, so for the 3rd one,
// to apply the clo rule to <a^x^x>p^[x_0, x_1], it must be the case that x_1 <= <a^x^x>p
    ↪ , thus
// <a^x><a^x^x>p <= <a^x^x>p, thus <a^x^x> must be a subterm of <a^x>, which it is not
test bool applyClo_test_29() {
        CloGSequent input = [term(\mod(dIter(dIter(atomG(agame("a")))), atomP(prop("p"))),
            ↪  [], false)];
        tuple[MaybeSequent, CloGName] output1 = <sequent([term(and(atomP(prop("p")), \mod(
            ↪ dIter(atomG(agame("a"))), \mod(dIter(dIter(atomG(agame("a")))), atomP(prop
            ↪ ("p"))))), [nameS("x", 0)], false)]), nameS("x", 0)>;
        CloSeqs closeqs1 = (nameS("x", 0): <[term(\mod(dIter(dIter(atomG(agame("a")))),
            ↪ atomP(prop("p"))), [], false)], 0>);
        tuple[MaybeSequent, CloGName] output2 = <sequent([term(and(\mod(dIter(dIter(atomG(
            ↪ agame("a")))), atomP(prop("p"))),\mod(atomG(agame("a")), \mod(dIter(atomG(
            ↪ agame("a"))), \mod(dIter(dIter(atomG(agame("a")))), atomP(prop("p")))))), [
            ↪ nameS("x", 0), nameS("x", 1)],false)]), nameS("x", 1)>;
        CloSeqs closeqs2 = (
                nameS("x", 0): <[term(\mod(dIter(dIter(atomG(agame("a")))), atomP(prop("p")
                    ↪ )), [], false)], 0>,
                nameS("x", 1): <[term(\mod(atomG(agame("a")), \mod(dIter(dIter(atomG(agame
                    ↪ ("a")))), atomP(prop("p")))), [], false)], 0>
        );
        tuple[MaybeSequent, CloGName] output3 = <noSeq(), name("")>;

        tuple[MaybeSequent, CloGName] result1 = applyClo(input, 0, ());
        MaybeSequents result2 = applyAnd(result1[0].seq, 0);
        tuple[MaybeSequent, CloGName] result3 = applyClo(result2.right, 0, closeqs1);
        MaybeSequents result4 = applyAnd(result3[0].seq, 0);
        tuple[MaybeSequent, CloGName] result5 = applyClo(result4.right, 0, closeqs2);

        return result1 == output1 && result3 == output2 && result5 == output3;
}
// <a||b>p^[]
test bool applyClo_test_31() {
        CloGSequent input = [term(\mod(choice(atomG(agame("a")), atomG(agame("b"))), atomP
            ↪ (prop("p"))), [], false)];
        tuple[MaybeSequent, CloGName] output = <noSeq(), name("")>;

        return applyClo(input, 0, ()) == output;
}
// <(a^x)*>p^[]
test bool applyClo_test_32() {
        CloGSequent input = [term(\mod(iter(dIter(atomG(agame("a")))), atomP(prop("p"))),
            ↪ [], false)];
        tuple[MaybeSequent, CloGName] output = <noSeq(), name("")>;

        return applyClo(input, 0, ()) == output;
}
```