Proof Transformations for Game Logic

Bachelor Thesis

Author: Christopher Worthington
First Supervisor: Helle Hvid Hansen
Second Supervisor: Tijs van der Storm

Abstract

This bachelor thesis will cover the implementation of a tool to perform a proof transformation from one proof systems to another in game logic. Game logic is a logic introduced by Rohit Parikh for reasoning about two-player determined games. The proof systems and transformation were introduced in a recent game logic completeness paper. The two proof systems, named CloG and G, are sequent calculi that use different axioms and inference rules to prove propositions in game logic. The tool successfully allows the user to input a proof of a proposition in CloG and retrieve a proof for the same proposition in G. The reason for this transformation is to investigate the relation between proofs in CloG and G when using this transformation and, due to there existing an efficient proof search for CloG, to perhaps be used in a proof search for G. The metaprogramming language Rascal is used to implement the transformation as its tools for using abstract syntax trees are well suited to handling proof trees and game logic expressions.

Contents

1 Introduction **5**

2 Game Logic Overview **5**
2.1 Game Logic Introduction . . . 5
2.2 Simple Game Example . . . 6
2.3 Game Logic Syntax . . . 7
2.4 Game Logic Semantics . . . 8
2.5 Calculating a Formula Valuation Within a Model . . . 9

3 Proof Systems and Transformation **12**
3.1 The Proof System CloG . . . 12
3.1.1 Example CloG Proof With One $\mathsf{clo_x}$ Application . . . 13
3.1.2 Example CloG Proof With Multiple $\mathsf{clo_x}$ Applications . . . 13
3.2 The Proof System G . . . 15
3.2.1 Example G Proof . . . 16
3.3 CloG to G Transformation . . . 17
3.3.1 Bullet Translation . . . 17
3.3.2 Direct Transformations Of CloG Rules To G Rules . . . 18
3.3.3 Transformation Of A $\mathsf{clo_x}$ Rule Application . . . 19
3.3.4 Transformation Of A Closure Rule Application . . . 20
3.3.5 Transformation Of An exp Rule Application . . . 20
3.3.6 Example Transformation . . . 21

4 Existing Literature on Proof Transformation Implementation **24**

5 Requirements for the Transformation Tool **25**
5.1 Must Have . . . 25
5.2 Should Have . . . 25
5.3 Could Have . . . 25
5.4 Non-Functional Requirements . . . 26

6 Implementation **26**
6.1 Development Technology . . . 26
6.2 Structure . . . 26
6.3 Abstract Syntax . . . 27
6.4 Implementing The CloG to G Transformation . . . 28
6.4.1 Implementing The Bullet Translation . . . 28
6.4.2 Mapping Rules to Derivations . . . 28
6.5 Interaction With the Tool . . . 29
6.5.1 CloG Input Syntax . . . 29
6.5.2 LaTeX output . . . 30
6.5.3 Main Function . . . 31

7 Testing **31**
7.1 Automated Testing . . . 31
7.2 Manual testing . . . 32

8 Results **32**

9 Conclusion **33**
9.1 Evaluation . . . 33
9.2 Future Work . . . 34

A Rascal Annotated Code **35**

1 Introduction

The concept of a formal deductive system consisting of axioms and inference rules was first designed by David Hilbert. This formalisation allows for proofs to be made into mathematical objects [1]. These proof systems can be defined for many different logics. If a system is sound and complete for a logic, this can be proven. It is often also the case that multiple proof systems exist for the same logic. Then if the same propositions can be proven by multiple proof systems it is also sometimes possible to define a transformation from one proof system to another.

The logic that this project will concern is game logic. Game logic is a monotonic modal logic that allows reasoning about what outcomes can be ensured by a player in two-player determined games, introduced by Rohit Parikh [2]. The first completeness proof for a game logic proof system was only recently created [3]. This paper proves completeness of multiple proof systems via proof transformations.

The chain of transformations in the paper start from the proof system Clo, an existing complete proof system for the modal $\mu-$calculus. The paper introduces three proof systems, CloM, CloG and G. CloM is another system for $\mu-$calculus. CloG and G are both proof systems for game logic. The transformations go from Clo to CloM, to CloG, to G, and finally to Par as shown in Figure 1.1. Par being the first proof system for game logic, also created by game logic's creator Parikh.

$$\mathsf{Par} \xleftarrow{1)} \mathsf{G} \xleftarrow{2)} \mathsf{CloG} \xleftarrow{3)} \mathsf{CloM} \xleftarrow{4)} \mathsf{Clo}$$

Figure 1.1: Transformations Introduced in [3]

There are currently no automated theorem provers for game logic, or any digital implementations for game logic in general. Therefore, it would be a valuable project to implement some of these proof systems and transformations. We know that there exists an efficient search strategy for CloG so transformations from that system would be most desirable, since we could find a proof in CloG and subsequently in other systems using the transformation. This leads to the following research question:

> How can we create a tool that can transform a proof formulated in CloG to a proof formulated in G?

We can use this tool to study the relation between the proofs in CloG and proofs in G with ease. We will be able to see how the proofs relate in in structure, and how they change in size and complexity. Proofs in Par are also more intuitive to read compared to proofs in CloG. We could use this tool in combination with the transformation from G to Par to find proofs in Par using the efficient proof search strategy for CloG. Though we don't know how large and complex these proofs might get.

2 Game Logic Overview

2.1 Game Logic Introduction

In this section I will introduce the concept of game logic, and explain its syntax and semantics. As described briefly in the introduction, game logic allows reasoning about strategic ability of players in determined two-player games [2]. These two players are usually referred to as either player one and player two or as Angel and Demon. The operators in the language are also sometimes named angelic or demonic so I will refer to the players as Angel and Demon in order to keep this more intuitive. To describe strategic ability, we use a modality of the form $\langle\gamma\rangle\varphi$. This modality says Angel has a strategy in game γ to ensure that the outcome satisfies the proposition φ. The proposition φ can take the form of any propositional formula and the $\langle\gamma\rangle$ is the unary modality. A modality can be used within any propositional formula and likewise could appear within φ.

2.2 Simple Game Example

As previously mentioned, the modality $\langle\gamma\rangle\varphi$ expresses that Angel has a strategy in the game γ to ensure the outcome satisfies φ. Let us take a look at a simple example for one of these games to visualise some possible strategies and outcomes for each player. We will call the game in Figure 2.1, a, and use atomic propositions p, q and r. At each turn in game a, a player can either go left or right. After two turns the outcome will be some state with some set of atomic propositions. One turn for Angel and then the next for demon. We can refer to the player's strategies with L, R, LL, LR, RL, or LR. The first two to describe Angel's strategies, L for choosing left and R for choosing right. Then the rest describe Demon's strategies based on whether Angel goes left or right respectively. So for example, if Demon's strategy is LR, then when Angel goes left so does Demon, and when Angel goes right Demon also goes right.

Based on what Angel does, Demon will have a limited number of outcomes they can ensure. Angel also can't know which move Demon will make so can also only ensure a certain number of outcomes. So in this example, $\langle a\rangle q$ is true since q will always be true if Angel uses strategy L. It is also true that Demon can ensure the outcome satisfies p with the strategy LR. If we want to express Demon's ability with an alternate modality, we need the property of *determinacy* to hold in a game. If this property holds in a game α then we have: a player can ensure some proposition φ is satisfied in α, if and only if the other player can not ensure $\neg\varphi$ is satisfied in α. In the case of this property holing, we can abbreviate the form $\neg\langle\alpha\rangle\neg\varphi$ to a logically equivalent modality $[\alpha]\varphi$. This abbreviation means Demon has a strategy in a game α to ensure the outcome satisfies φ. We can see that the game a is *determined* as all moves will result in a guaranteed next state. So if a player is able to ensure an outcome, then the other player has no strategy to ensure that outcome doesn't happen.

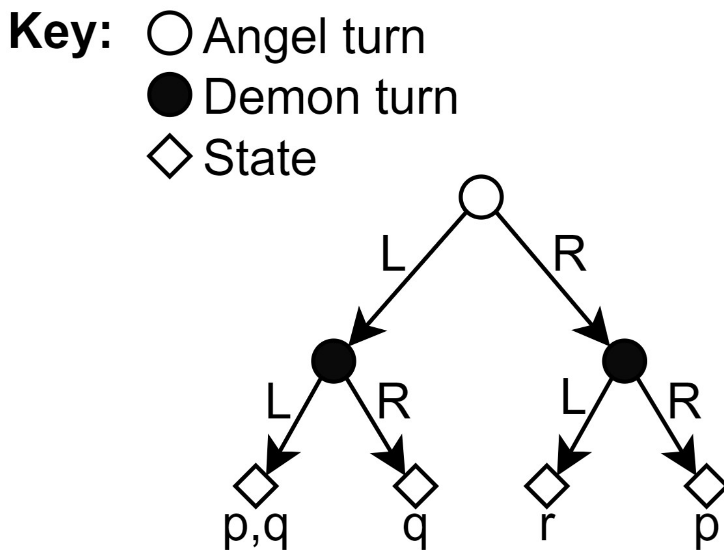

Figure 2.1: Simple game a, with atomic propositions

Another important property that must hold for games in game logic is *Monotonicity.* In order for Monotonicity to hold in a, $\langle a\rangle\varphi \to \langle a\rangle(\varphi \vee \psi)$ must hold for any propositions φ and ψ. We can see this holds in game a as if Angel has a strategy to ensure some set of outcomes which satisfy φ, then those outcomes will also always have $\varphi \vee \psi$ no matter what ψ is. The game a is, however, not *disjunctive* as we do not have $\langle a\rangle(\varphi \vee \psi) \to \langle a\rangle\varphi$. A counter example for this in a would be that we have $\langle a\rangle(r \vee p)$ with strategy R, but not $\langle a\rangle r$ or $\langle a\rangle p$.

To explain some further terminology, let us look at the same game a but in Figure 2.2 with named outcome states. We define a player *effective* for some set of states U in a game α if the player has a strategy to ensure

the outcome state of game α is in U. So for example, in game a, Angel is effective for the set $U_0 = \{x_1, x_2\}$ with strategy L. This is not to say that Angel can choose to ensure the outcome is x_1 or choose to ensure the outcome is x_2, but that Angel can ensure the outcome is at least in the set U_0. Angel is also effective for any superset U_1 of U_0. This is because Angel can ensure the outcome to be in the set U_0, no matter what states are in $U_1 \backslash U_0$, and subsequently also in U_1. This is another property that holds in all *monotonic* games.

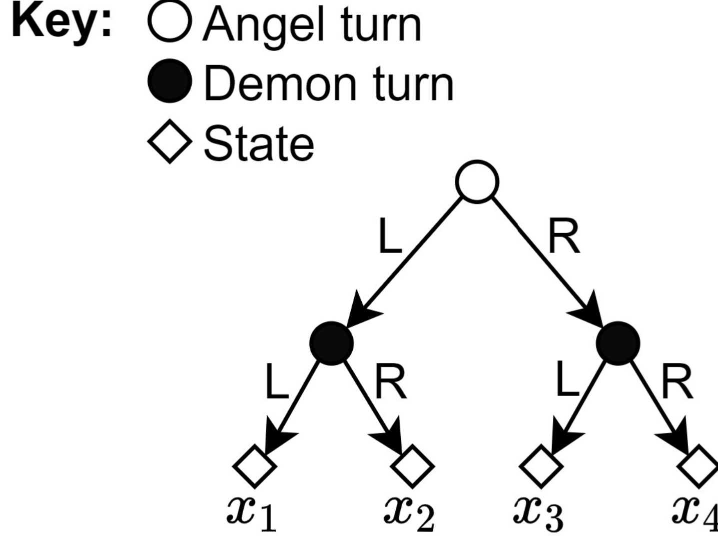

Figure 2.2: Simple game a, with labelled outcome states

2.3 Game Logic Syntax

For the syntax of game logic, there are two types of term. These are either games or propositions. We assume that given are a set of atomic propositions Φ_0 and a set of atomic determined monotonic games Γ_0. The syntactic form of propositions φ and games γ is defined as follows:

Definition 2.1 (Game Logic Syntax)**.**

$$\begin{aligned} \varphi &:= \bot \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\gamma\rangle\varphi \\ \gamma &:= g \mid \varphi? \mid \gamma;\gamma \mid \gamma \sqcup \gamma \mid \gamma^* \mid \gamma^d \end{aligned}$$

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

Let us give some explanation of the game operators. I have already introduced the modality $\langle\gamma\rangle\varphi$. A sequential game concatenation, $\gamma_1;\gamma_2$, is the game where first γ_1 is played followed by γ_2. An angelic choice, $\gamma_1 \sqcup \gamma_2$, is the game where Angel chooses to either play γ_1 or γ_2. An angelic test game, $\varphi?$, consists of checking if φ is true at the current state, and if it is not true then Angel immediately loses. It can be assumed Angel will never choose an option to lose if possible, so the test game is used to cause Angel to not be able to make certain moves at certain states where some φ is not true. An angelic iteration, γ^*, is the game where Angel can choose to play γ any number of times (this includes zero times). Angel does not first choose how many and then play that number, instead, Angel chooses at the start whether play γ and after each time playing γ, can choose to play again or stop and move on. The dual operator, γ^d, is the game γ but with the player' roles reversed. Any move made by Angel in γ is instead made by Demon in γ^d and vice versa.

Additionally we can have the usual propositional abbreviations $\wedge$, $\rightarrow$ and $\leftrightarrow$. We can also have the $[\gamma]\varphi$ abbreviation as described in the previous section. Some more abbreviations for the demonic alternatives of the game operators are also possible:

Definition 2.2 (Demonic Operators)**.**

$$\gamma_1 \sqcap \gamma_2 \leftrightarrow (\gamma_1^d \sqcup \gamma_2^d)^d \qquad \varphi! \leftrightarrow ((\neg\varphi)?)^d \qquad \gamma^\times \leftrightarrow ((\gamma^d)^*)^d$$

These being demonic choice, demonic test, and demonic iteration respectively. In the case of a demonic choice, Demon can choose which of the two games to play. In a demonic test, Angel immediately wins if φ is true. In a demonic iteration, Demon chooses how many times to play the game, in the same way as Angel would in an angelic iteration.

Definition 2.3. We also often refer to any modality with an iteration of the form $\langle\gamma^*\rangle\varphi$ or $\langle\gamma^x\rangle\varphi$ as a *fixed point formula* and this will be important terminology when discussing the proof systems and transformations later in this project.

In [3] there is also a syntactic normal form language $\mathcal{L}_{\text{NF}}$ defined. In this normal form, both negation and dual are only applied to atomic propositions and atomic games respectively. The syntactic definition of $\mathcal{L}_{\text{NF}}$ is as follows:

Definition 2.4 (Game Logic Syntax Normal Form)**.**

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

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

2.4 Game Logic Semantics

For the semantics of game logic we need a defined model to describe the states, atomic propositions and atomic games for a given example. A game model is given in the form $\mathbb{S} = (S, \{E_g \mid g \in \Gamma_0\}, V)$ consisting of a set of states S, a set containing an effectivity function for each atomic game $\{E_g \mid g \in \Gamma_0\}$, and a valuation function V that maps each atomic proposition to a set of all states at which they are true. Each effectivity function E_g maps any set of states U_0 to a set of all states at which Angel is effective in game g for U_0.

We also need some further function to extend the atomic valuation to valuate complex terms. I will use the function $[\![\varphi]\!]^{\mathbb{S}}$ for the valuation of complex formulae φ in model $\mathbb{S}$ and the effectivity function $E_\gamma(X)$ for complex games γ on a set U, as defined in [3] on the language $\mathcal{L}_{\text{NF}}$:

Definition 2.5 (Semantic Valuation for Complex Formulae)**.**

$$\begin{aligned} [\![p]\!]^{\mathbb{S}} &:= V(p) \quad \text{(Atomic valuation given in } \mathbb{S}) \\ [\![\neg\varphi]\!]^{\mathbb{S}} &:= S \backslash [\![\varphi]\!]^{\mathbb{S}} \\ [\![\varphi \vee \psi]\!]^{\mathbb{S}} &:= [\![\varphi]\!]^{\mathbb{S}} \cup [\![\psi]\!]^{\mathbb{S}} \\ [\![\varphi \wedge \psi]\!]^{\mathbb{S}} &:= [\![\varphi]\!]^{\mathbb{S}} \cap [\![\psi]\!]^{\mathbb{S}} \\ [\![\langle\gamma\rangle\varphi]\!]^{\mathbb{S}} &:= E_\gamma([\![\varphi]\!]^{\mathbb{S}}) \\ E_g(X) & \qquad \text{(Atomic effectivity function given in } \mathbb{S}) \\ E_{(\gamma^d)}(X) &:= S \backslash E_\gamma(S \backslash X) \\ E_{\gamma;\delta}(X) &:= E_\gamma(E_\delta(X)) \\ E_{\gamma \sqcup \delta}(X) &:= E_\gamma(X) \cup E_\delta(X) \\ E_{\gamma \sqcap \delta}(X) &:= E_\gamma(X) \cap E_\delta(X) \\ E_{\gamma^*}(X) &:= \text{lfp } Y.X \cup E_\gamma(Y) \\ E_{\gamma^\times}(X) &:= \text{gfp } Y.X \cap E_\gamma(Y) \\ E_{\varphi?}(X) &:= [\![\varphi]\!]^{\mathbb{S}} \cap X \\ E_{\varphi!}(X) &:= [\![\varphi]\!]^{\mathbb{S}} \cup X \end{aligned}$$

Most operations in these functions are standard set theory, however, the least and greatest fixed point operations, for γ^* and $\gamma^\times$, require more complex calculations. For each fixed point calculation, we construct a function which takes and returns a set of states, and this function is formulated from the patterns above following either lfp Y or gfp Y. The function is in terms of a new arbitrary set Y. Once this function is constructed, we either input the full set of states or the empty set depending on if we are calculating an lfp or a gfp respectively. Finally we iteratively check if the output of the function is the same as the input, if not, then we put the output back into the function and repeat, if it is, then we have arrived at our fixed point set of states.

2.5 Calculating a Formula Valuation Within a Model

Using the semantics of game logic, we can calculate the valuation for a given game formula φ at a given state. One reason we may want to calculate a valuation is to show that a formula φ is not a tautology via a counter example game model $\mathbb{S}$. Take this propositional formula:

Example 2.1 (Game Logic Formula)**.**

$$\varphi = \langle((p?;a^\times) \sqcup (\neg p?;b^\times))^\times\rangle q \to \langle(a^\times \sqcap b^\times)^\times\rangle q$$

We can show that this formula is not a tautology by defining a game model and calculating the valuation of the formula within the model. If not all states are in the valuation then clearly the formula is not always true at any state in any model. For this we have the following example model which takes the form $\mathbb{S} = (S, \{E_g \mid g \in \Gamma_0\}, V)$:

Example 2.2 (Game Model)**.**

$$\begin{aligned}
S &= \{s_1, s_2\} \\
\\
E_a(\emptyset) &= \emptyset \\
E_a(\{s_1\}) &= \{s_1, s_2\} \\
E_a(\{s_2\}) &= \emptyset \\
E_a(\{s_1, s_2\}) &= \{s_1, s_2\} \\
E_b(\emptyset) &= \emptyset \\
E_b(\{s_1\}) &= \emptyset \\
E_b(\{s_2\}) &= \{s_1, s_2\} \\
E_b(\{s_1, s_2\}) &= \{s_1, s_2\} \\
\\
V(a) &= \{s_1, s_2\} \\
V(q) &= \{s_1\}
\end{aligned}$$

For consistency, I will also convert the formula to $\mathcal{L}_{\text{NF}}$ before calculating the valuation:

$$\begin{aligned}
\varphi &= \langle((p?;a^\times) \sqcup (\neg p?;b^\times))^\times\rangle q \to \langle(a^\times \sqcap b^\times)^\times\rangle q \\
&= \neg\langle((p?;a^\times) \sqcup (\neg p?;b^\times))^\times\rangle q \lor \langle(a^\times \sqcap b^\times)^\times\rangle q \\
&= \langle((p?;a^\times) \sqcup (\neg p?;b^\times))^{\times^d}\rangle \neg q \lor \langle(a^\times \sqcap b^\times)^\times\rangle q \\
&= \langle(((p?;a^\times) \sqcup (\neg p?;b^\times))^d)^*\rangle \neg q \lor \langle(a^\times \sqcap b^\times)^\times\rangle q \\
&= \langle((p?;a^\times)^d \sqcap (\neg p?;b^\times)^d)^*\rangle \neg q \lor \langle(a^\times \sqcap b^\times)^\times\rangle q \\
&= \langle((p?^d;a^{\times^d}) \sqcap (\neg p?^d;b^{\times^d}))^*\rangle \neg q \lor \langle(a^\times \sqcap b^\times)^\times\rangle q \\
&= \langle((\neg p!;a^{d^*}) \sqcap (p!;b^{d^*}))^*\rangle \neg q \lor \langle(a^\times \sqcap b^\times)^\times\rangle q
\end{aligned}$$

Now we can start the calculation of the valuation:

$$
\begin{aligned}
[\![\varphi]\!]^{\mathbb{S}} &= [\![\langle((\neg p!; a^{d^*}) \sqcap (p!; b^{d^*}))^*\rangle\neg q \vee \langle(a^\times \sqcap b^\times)^\times\rangle q]\!]^{\mathbb{S}} && (2.1)\\
&= [\![\langle((\neg p!; a^{d^*}) \sqcap (p!; b^{d^*}))^*\rangle\neg q]\!]^{\mathbb{S}} \cup [\![\langle(a^\times \sqcap b^\times)^\times\rangle q]\!]^{\mathbb{S}} && (2.2)\\
&= E_{((\neg p!; a^{d^*}) \sqcap (p!; b^{d^*}))^*}([\![\neg q]\!]^{\mathbb{S}}) \cup E_{(a^\times \sqcap b^\times)^\times}([\![q]\!]^{\mathbb{S}}) && (2.3)\\
&= (\text{lfp } Y.[\![\neg q]\!]^{\mathbb{S}} \cup E_{(\neg p!; a^{d^*}) \sqcap (p!; b^{d^*})}(Y)) \cup (\text{gfp } Y.[\![q]\!]^{\mathbb{S}} \cap E_{a^\times \sqcap b^\times}(Y)) && (2.4)
\end{aligned}
$$

We have two fixed points to calculate which means we need to formulate two functions. The first is $f(Y)$ for the least fixed point operation in equation line 2.4.

$$
\begin{aligned}
\text{let } f(Y) &= [\![\neg q]\!]^{\mathbb{S}} \cup E_{(\neg p!; a^{d^*}) \sqcap (p!; b^{d^*})}(Y) && (2.5)\\
&= (S \backslash [\![q]\!]^{\mathbb{S}}) \cup (E_{\neg p!; a^{d^*}}(Y) \cap E_{p!; b^{d^*}}(Y)) && (2.6)\\
&= (S \backslash [\![q]\!]^{\mathbb{S}}) \cup (E_{\neg p!; a^{d^*}}(Y) \cap E_{p!; b^{d^*}}(Y)) && (2.7)\\
&= (\{s_1, s_2\} \backslash V(q)) \cup (E_{\neg p!}(E_{a^{d^*}}(Y)) \cap E_{p!}(E_{b^{d^*}}(Y))) && (2.8)\\
&= (\{s_1, s_2\} \backslash \{s_1\}) \cup (([\![\neg p]\!]^{\mathbb{S}} \cup E_{a^{d^*}}(Y)) \cap ([\![p]\!]^{\mathbb{S}} \cup E_{b^{d^*}}(Y))) && (2.9)\\
&= \{s_2\} \cup (((S \backslash [\![p]\!]^{\mathbb{S}}) \cup (\text{lfp } Z.Y \cup E_{a^d}(Z))) \cap (V(p) \cup (\text{lfp } Z.Y \cup E_{b^d}(Z)))) && (2.10)\\
&= \{s_2\} \cup (((\{s_1, s_2\} \backslash V(p)) \cup (\text{lfp } Z.Y \cup E_{a^d}(Z))) \cap (\{s_1, s_2\} \cup (\text{lfp } Z.Y \cup E_{b^d}(Z)))) && (2.11)\\
&= \{s_2\} \cup (((\{s_1, s_2\} \backslash \{s_1, s_2\}) \cup (\text{lfp } Z.Y \cup E_{a^d}(Z))) \cap \{s_1, s_2\})) && (2.12)\\
&= \{s_2\} \cup (\text{lfp } Z.Y \cup E_{a^d}(Z)) && (2.13)
\end{aligned}
$$

The next is $g(Y)$ for the greatest fixed point operation in equation line 2.4

$$
\begin{aligned}
\text{let } g(Y) &= [\![q]\!]^{\mathbb{S}} \cap E_{a^\times \sqcap b^\times}(Y)) && (2.14)\\
&= (\{s_1\} \cap E_{a^\times}(Y) \cap E_{b^\times}(Y) && (2.15)\\
&= \{s_1\} \cap (\text{gfp } Z.Y \cap E_a(Z)) \cap (\text{gfp } Z.Y \cap E_b(Z)) && (2.16)
\end{aligned}
$$

Within these two functions we also have some nested fixed points to calculate. The function $h(Y, Z)$ for the least fixed point operation on equation line 2.13 is formulated as follows:

$$
\begin{aligned}
\text{let } h(Y, Z) &= Y \cup E_{a^d}(Z) && (2.17)\\
&= Y \cup (S \backslash E_a(S \backslash Z)) && (2.18)\\
&= Y \cup (\{s_1, s_2\} \backslash E_a(\{s_1, s_2\} \backslash Z)) && (2.19)
\end{aligned}
$$

The other monotone functions from equation line 2.16 are already simplified as much as they can be:

$$
\begin{aligned}
\text{let } w(Y, Z) = Y \cap E_a(Z) && (2.20)\\
\text{let } v(Y, Z) = Y \cap E_b(Z) && (2.21)
\end{aligned}
$$

Now we calculate the fixed points by iterations:

- lfp $Y.f(Y)$ from equation line 2.4:
  Iteration 0: $f(\emptyset) = \{s_2\} \cup (\text{lfp } Z.h(\emptyset, Z))$
  - lfp $Z.h(\emptyset, Z)$ from equation line 2.13:
    Iteration 0: $h(\emptyset, \emptyset) = \emptyset \ \cup \ (\{s_1, s_2\} \backslash E_a(\{s_1, s_2\} \backslash \emptyset)) = \{s_1, s_2\} \backslash E_a(\{s_1, s_2\}) = \{s_1, s_2\} \backslash \{s_1, s_2\} = \emptyset$
    $\Rightarrow$ lfp $Z.h(\emptyset, Z) = \emptyset$

$\Rightarrow f(\emptyset) = \{s_2\} \cup \emptyset = \{s_2\}$
Iteration 1: $f(\{s_2\}) = \{s_2\} \cup (\text{lfp } Z.h(\{s_2\}, Z))$

- lfp $Z.h(\{s_2\}, Z)$ from equation line 2.13:
  Iteration 0: $h(\{s_2\}, \emptyset) = \{s_2\} \cup (\{s_1, s_2\} \backslash E_a(\{s_1, s_2\} \backslash \emptyset)) = \{s_2\} \cup (\{s_1, s_2\} \backslash E_a(\{s_1, s_2\})) = \{s_2\} \cup (\{s_1, s_2\} \backslash \{s_1, s_2\}) = \{s_2\} \cup \emptyset = \{s_2\}$
  Iteration 1: $h(\{s_2\}, \{s_2\}) = \{s_2\} \cup (\{s_1, s_2\} \backslash E_a(\{s_1, s_2\} \backslash \{s_2\})) = \{s_2\} \cup (\{s_1, s_2\} \backslash E_a(\{s_1\})) = \{s_2\} \cup (\{s_1, s_2\} \backslash \{s_1, s_2\}) = \{s_2\} \cup \emptyset = \{s_2\}$
  $\Rightarrow$ lfp $Z.h(\{s_2\}, Z) = \{s_2\}$

$\Rightarrow f(\{s_2\}) = \{s_2\} \cup \{s_2\} = \{s_2\}$
$\Rightarrow$lfp $Y.f(Y) = \{s_2\}$

- gfp $Y.g(Y)$ from equation line 2.4:
  Iteration 0: $g(\{s_1, s_2\}) = \{s_1\} \cap (\text{gfp } Z.w(\{s_1, s_2\}, Z)) \cap (\text{gfp } Z.v(\{s_1, s_2\}, Z))$

  - gfp $Z.w(\{s_1, s_2\}, Z)$ from equation line 2.16:
    Iteration 0: $w(\{s_1, s_2\}, \{s_1, s_2\}) = \{s_1, s_2\} \cap E_a(\{s_1, s_2\}) = \{s_1, s_2\} \cap \{s_1, s_2\} = \{s_1, s_2\}$
    $\Rightarrow$ gfp $Z.w(\{s_1, s_2\}, Z) = \{s_1, s_2\}$
  - gfp $Z.v(\{s_1, s_2\}, Z)$ from equation line 2.16:
    Iteration 0: $v(\{s_1, s_2\}, \{s_1, s_2\}) = \{s_1, s_2\} \cap E_b(\{s_1, s_2\}) = \{s_1, s_2\} \cap \{s_1, s_2\} = \{s_1, s_2\}$
    $\Rightarrow$ gfp $Z.v(\{s_1, s_2\}, Z) = \{s_1, s_2\}$

  $\Rightarrow g(\{s_1, s_2\}) = \{s_1\} \cap \{s_1, s_2\} \cap \{s_1, s_2\} = \{s_1\}$
  Iteration 1: $g(\{s_1\}) = \{s_1\} \cap (\text{gfp } Z.w(\{s_1\}, Z)) \cap (\text{gfp } Z.v(\{s_1\}, Z))$

  - gfp $Z.w(\{s_1\}, Z)$ from equation line 2.16:
    Iteration 0: $w(\{s_1\}, \{s_1, s_2\}) = \{s_1\} \cap E_a(\{s_1, s_2\}) = \{s_1\} \cap \{s_1, s_2\} = \{s_1\}$
    Iteration 1: $w(\{s_1\}, \{s_1\}) = \{s_1\} \cap E_a(\{s_1\}) = \{s_1\} \cap \{s_1, s_2\} = \{s_1\}$
    $\Rightarrow$ gfp $Z.w(\{s_1\}, Z) = \{s_1\}$
  - gfp $Z.v(\{s_1\}, Z)$ from equation line 2.16:
    Iteration 0: $v(\{s_1\}, \{s_1, s_2\}) = \{s_1\} \cap E_b(\{s_1, s_2\}) = \{s_1\} \cap \{s_1, s_2\} = \{s_1\}$
    Iteration 1: $v(\{s_1\}, \{s_1\}) = \{s_1\} \cap E_b(\{s_1\}) = \{s_1\} \cap \emptyset = \emptyset$
    Iteration 2: $v(\{s_1\}, \emptyset) = \{s_1\} \cap E_b(\emptyset) = \{s_1\} \cap \emptyset = \emptyset$
    $\Rightarrow$ gfp $Z.v(\{s_1\}, Z) = \emptyset$

  $\Rightarrow g(\{s_1\}) = \{s_1\} \cap \{s_1\} \cap \emptyset = \emptyset$
  Iteration 2: $g(\emptyset) = \{s_1\} \cap (\text{gfp } Z.w(\emptyset, Z)) \cap (\text{gfp } Z.v(\emptyset, Z))$

  - gfp $Z.w(\emptyset, Z)$ from equation line 2.16:
    Iteration 0: $w(\emptyset, \{s_1, s_2\}) = \emptyset \cap E_a(\{s_1, s_2\}) = \emptyset$
    Iteration 1: $w(\emptyset, \emptyset) = \emptyset \cap E_a(\emptyset) = \emptyset$
    $\Rightarrow$ gfp $Z.w(\emptyset, Z) = \emptyset$
  - gfp $Z.v(\emptyset, Z)$ from equation line 2.16:
    Iteration 0: $v(\emptyset, \{s_1, s_2\}) = \emptyset \cap E_b(\{s_1, s_2\}) = \emptyset$
    Iteration 1: $v(\emptyset, \emptyset) = \emptyset \cap E_b(\emptyset) = \emptyset$
    $\Rightarrow$ gfp $Z.v(\emptyset, Z) = \emptyset$

  $\Rightarrow f(\emptyset) = \{s_1\} \cap \emptyset \cap \emptyset = \emptyset$
  $\Rightarrow$ gfp $Y.g(Y) = \emptyset$

Now that we have calculated the fixed points we can get our final valuation:

$$\begin{aligned} [\![\varphi]\!]^{\mathbb{S}} &= (\text{lfp } Y.f(Y)) \cup (\text{gfp } Y.g(Y)) \\ &= \{s_2\} \cup \emptyset \\ &= \{s_2\} \end{aligned}$$

Here we can see that $s_1 \notin [\![\varphi]\!]^{\mathbb{S}}$ so φ does not evaluate to true at s_1 in this model and therefore φ is not a tautology.

3 Proof Systems and Transformation

Both proof systems that this project concerns are sequent calculi. These are different from Hilbert style proof systems as instead of one formula per line of reasoning, there is instead a sequent of the form $A_1, ..., A_n \vdash B_1, ..., B_m$ where A_i and B_i are propositional formulas [1]. Each sequent is interpreted as $(A_1 \wedge ... \wedge A_n) \rightarrow (B_1 \vee ... \vee B_m)$. These proof systems each consist of a set of axioms and a set of inference rules. Axioms are some tautological patterns that can be introduced within a proof with no derivation, and inference rules are a pattern of derivation from some line of reasoning to another. Hilbert systems tend to have many axioms and few inference rules, whereas sequent calculi usually have many rules and just a few axioms. These sequent proofs are generally read analytically from the root of the tree up to the branches. This is also how proofs are effiently searched for in these proof systems. Therefore, this is how I will refer to the proofs along with their rules and axioms throughout this project.

3.1 The Proof System CloG

An important feature of the CloG proof system is the labelling system. Each formula within a sequent is labelled with a list of names. Each of these names uniquely refers to a fixed point formula of the form introduced in Definition 2.3. These names are used to detect a repeated context when unfolding a fixed point formula, this is further explained later in this section.

A partial order $\preceq$ is also defined on the fixed point formulas:

Definition 3.1. $\langle\gamma^{\circ}\rangle\varphi \prec \langle\delta^{\dagger}\rangle\psi$ with $\circ, \dagger \in \{*, \times\}$ if and only if $\delta^{\dagger}$ is a subterm of γ°.

The partial order $\preceq$ is the less strict version with equality. Names also inherit this partial order from the fixed point formulas they are names for:

Definition 3.2. Given names, x_0 and x_1, of fixed point formulas φ and ψ respectively, $\mathsf{x}_0 \preceq \mathsf{x}_1$ if and only if $\varphi \preceq \psi$.

The partial order can also be used with a list of names in a label:

Definition 3.3. given a label, a, and a fixed point formula φ, then $\mathsf{a} \preceq \varphi$ if and only if for every fixed point formula ψ referred to by some name in a, $\psi \preceq \varphi$.

CloG sequents take the form $(A_1)^{\mathsf{a}_1}, ..., (A_n)^{\mathsf{a}_n}$ where A_i is a game logic formula in the language $\mathcal{L}_{\mathrm{NF}}$, and a_i is a label consisting of a list of names. The sequent is interpreted as $A_1 \vee ... \vee A_n$ and can be seen as a special case of the general form sequent introduced at the start of this section. A propositional formula φ is proven in a valid CloG proof if we have the sequent $(\varphi)^{\epsilon}$ at the root of the proof, and all branches are either closed or lead to an axiom.

The rules and axioms of CloG are shown in Figure 3.1. The presented inference rules show the patterns the active formulas take before and after a rule application along with the other formulas in the sequent represented by a Φ symbol. Ax1 is the axiom that can be used in the case of a sequent containing just an atomic proposition and its negation, both with empty labels. The $*$ rule application allows a sequent of the form $\Phi, (\langle\gamma^*\rangle\varphi)^{\mathsf{a}}$ below, and a sequent of the form $\Phi, (\varphi \vee \langle\gamma\rangle\langle\gamma^*\rangle\varphi)^{\mathsf{a}}$ above the rule application. The $*$ rule, however, has an additional side condition where the rule can only be applied if $a \preceq \langle\gamma^*\rangle\varphi$ holds according to Definition 3.3. The $\times$ rule has the same side condition as the $*$ rule but instead with a demonic iteration operator. The $\mathrm{clo}_{\mathsf{x}}$ rule is applied to a sequent of the form $\Phi, (\langle\gamma^{\times}\rangle\varphi)^{\mathsf{a}}$ and the demonic fixed point formula is *unfolded* in the application of the rule. The rule also introduces a new name x for the fixed point formula $\langle\gamma^{\times}\rangle\varphi$. The side condition here ensures that the new name does not already appear in the label a, or the label of any of the formulas in Φ, and also that $a \preceq \langle\gamma^{\times}\rangle\varphi$ again holds. In addition to the $\mathrm{clo}_{\mathsf{x}}$, we have a closure rule above. We can close a branch in the case that we have a sequent of the form $\Phi, (\langle\gamma^{\times}\rangle\varphi)^{\mathsf{a}\mathsf{x}}$ and

when the clo_x rule was applied, the sequent had the same form $\Phi, (\langle \gamma^\times \rangle \varphi)^\mathsf{a}$ without the added name x.

$$\dfrac{}{p^\varepsilon, (\neg p)^\varepsilon}\,\mathsf{Ax1} \qquad \dfrac{\varphi^\mathsf{a}, \psi^\mathsf{b}}{(\langle g \rangle \varphi)^\mathsf{a}, (\langle g^d \rangle \psi)^\mathsf{b}}\,\mathsf{mod}_m \qquad (\mathsf{a} \preccurlyeq \langle \gamma^* \rangle \varphi)\ \dfrac{\Phi, (\varphi \vee \langle \gamma \rangle \langle \gamma^* \rangle \varphi)^\mathsf{a}}{\Phi, (\langle \gamma^* \rangle \varphi)^\mathsf{a}}\,* \qquad \dfrac{\Phi, (\psi \wedge \varphi)^\mathsf{a}}{\Phi, (\langle \psi? \rangle \varphi)^\mathsf{a}}\,?$$

$$\dfrac{\Phi, \varphi^\mathsf{a}, \psi^\mathsf{a}}{\Phi, (\varphi \vee \psi)^\mathsf{a}}\,\vee \qquad \dfrac{\Phi, \varphi^\mathsf{a} \qquad \Phi, \psi^\mathsf{a}}{\Phi, (\varphi \wedge \psi)^\mathsf{a}}\,\wedge \qquad (\mathsf{a} \preccurlyeq \langle \gamma^\times \rangle \varphi)\ \dfrac{\Phi, (\varphi \wedge \langle \gamma \rangle \langle \gamma^\times \rangle \varphi)^\mathsf{a}}{\Phi, (\langle \gamma^\times \rangle \varphi)^\mathsf{a}}\,\times \qquad \dfrac{\Phi, (\psi \vee \varphi)^\mathsf{a}}{\Phi, (\langle \psi! \rangle \varphi)^\mathsf{a}}\,!$$

$$\dfrac{\Phi, (\langle \gamma \rangle \varphi \vee \langle \delta \rangle \varphi)^\mathsf{a}}{\Phi, (\langle \gamma \sqcup \delta \rangle \varphi)^\mathsf{a}}\,\sqcup \qquad \dfrac{\Phi, (\langle \gamma \rangle \varphi \wedge \langle \delta \rangle \varphi)^\mathsf{a}}{\Phi, (\langle \gamma \sqcap \delta \rangle \varphi)^\mathsf{a}}\,\sqcap$$

$$\dfrac{\Phi}{\Phi, \varphi^\mathsf{a}}\,\mathsf{weak} \qquad \dfrac{\Phi, \varphi^\mathsf{ab},}{\Phi, \varphi^\mathsf{axb}}\,\mathsf{exp} \qquad \dfrac{\Phi, (\langle \gamma \rangle \langle \delta \rangle \varphi)^\mathsf{a}}{\Phi, (\langle \gamma ; \delta \rangle \varphi)^\mathsf{a}}\,; \qquad (\mathsf{a} \preccurlyeq \mathsf{x} \in N_{\langle \gamma^\times \rangle \varphi}, \mathsf{x} \notin \Phi, \mathsf{a})\ \dfrac{\begin{matrix}[\Phi, \langle \gamma^\times \rangle \varphi^{\mathsf{ax}}]^\mathsf{x} \\ \vdots \\ \Phi, (\varphi \wedge \langle \gamma \rangle \langle \gamma^\times \rangle \varphi)^{\mathsf{ax}}\end{matrix}}{\Phi, (\langle \gamma^\times \rangle \varphi)^\mathsf{a}}\,\mathsf{clo}_\mathsf{x}$$

Figure 3.1: Axioms and rules of CloG

3.1.1 Example CloG Proof With One clo_x Application

Let us look at an example CloG proof for the tautological formula $\varphi = \langle (a \sqcap b)^\times \rangle p \to \langle (a^*; b)^\times \rangle p$. We need this formula in its form in the language $\mathcal{L}_{\mathrm{NF}}$ which is $\langle (a^d \sqcup b^d)^* \rangle \neg p \vee \langle (a^*; b)^\times \rangle p$. Then we start the proof search with this formula and an empty label $(\varphi)^\varphi$ at the root of the proof. Within the example proofs in this paper, the active formulas in each sequent, i.e. the formulas that are changed before and after a rule application, are highlighed in *red*.

Right branch (top part):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[(\langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (\langle (a^*; b)^\times \rangle p)^{x_0}]^{x_0}}{(\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (\langle b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}\,\mathrm{mod}_m}{(\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (\langle b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}, (\langle a \rangle \langle a^* \rangle \langle b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}\,\mathrm{weak}}{(\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (\langle b \rangle \langle (a^*; b)^\times \rangle p \vee \langle a \rangle \langle a^* \rangle \langle b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}\,\vee}{(\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (\langle a^* \rangle \langle b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}\,*}{(\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (\langle a^*; b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}\,;}{(\neg p)^\varepsilon, (\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (\langle a^*; b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}\,\mathrm{weak}$$

Left branch:

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

Lower part (the two branches above are the premises of $\wedge$):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\neg p)^\varepsilon, (\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (p)^{x_0} \qquad (\neg p)^\varepsilon, (\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (\langle a^*; b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}{(\neg p)^\varepsilon, (\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (p \wedge \langle a^*; b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}\,\wedge}{(\neg p)^\varepsilon, (\langle a^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (p \wedge \langle a^*; b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}\,\mathrm{weak}}{(\neg p)^\varepsilon, (\langle a^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p \vee \langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (p \wedge \langle a^*; b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}\,\vee}{(\neg p)^\varepsilon, (\langle a^d \sqcup b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (p \wedge \langle a^*; b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}\,\sqcup}{(\neg p \vee \langle a^d \sqcup b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (p \wedge \langle a^*; b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}\,\vee}{(\langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (p \wedge \langle a^*; b \rangle \langle (a^*; b)^\times \rangle p)^{x_0}}\,*}{(\langle (a^d \sqcup b^d)^* \rangle \neg p)^\varepsilon, (\langle (a^*; b)^\times \rangle p)^\varepsilon}\,\mathrm{clo}_{x_0}}{(\langle (a^d \sqcup b^d)^* \rangle \neg p \vee \langle (a^*; b)^\times \rangle p)^\varepsilon}\,\vee}{}$$

In this proof we used one application of the clo_x rule with the name x_0. The right hand branch is then closed at the end when there is the same context but with x_0 in the label.

3.1.2 Example CloG Proof With Multiple clo_x Applications

Now let us look at a larger proof for a formula $\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q \vee \langle ((p?; a^\times) \sqcup (\neg p?; b^\times))^\times \rangle q$, which will use more applications of the clo_x rule. In order to fit this proof onto the page, it is split into multiple parts. These parts refer to each other via $*$'s with subscript numeration.

$$[(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle a^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_2}]^{x_2}$$
$$\overline{(\langle a^d\rangle\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle a\rangle\langle a^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_2}}\ \text{mod}_m$$
$$\overline{(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle a^d\rangle\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle a\rangle\langle a^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_2}}\ \text{weak}$$
$$\overline{(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q\vee\langle a^d\rangle\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle a\rangle\langle a^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_2}}\ \vee$$
$$\overline{(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle a\rangle\langle a^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_2}}\ *$$
$$*_5$$

$$[(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0}]^{x_0}$$
$$\overline{(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_2}}\ \text{exp}$$
$$\overline{(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle a^d\rangle\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_2}}\ \text{weak}$$
$$\overline{(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q\vee\langle a^d\rangle\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_2}}\ \vee$$
$$\overline{(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_2}}\ *\qquad *_5$$
$$\overline{(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q\wedge\langle a\rangle\langle a^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_2}}\ \wedge$$
$$\overline{(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle a^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0}}\ \text{clo}_{x_2}$$
$$\overline{(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle a^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0}}\ \text{weak}$$
$$\overline{(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle a^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0}, (\neg p\wedge\langle b^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0}}\ \text{weak}$$
$$*_1$$

$$[(\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_1}]^{x_1}$$
$$\overline{(\langle b^d\rangle\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b\rangle\langle b^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_1}}\ \text{mod}_m$$
$$\overline{(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b^d\rangle\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b\rangle\langle b^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_1}}\ \text{weak}$$
$$\overline{(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q\vee\langle b^d\rangle\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b\rangle\langle b^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_1}}\ \vee$$
$$\overline{(\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b\rangle\langle b^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_1}}\ *$$
$$*_4$$

$$[(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0}]^{x_0}$$
$$\overline{(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b^d\rangle\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0}}\ \text{weak}$$
$$\overline{(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b^d\rangle\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_1}}\ \text{exp}$$
$$\overline{(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q\vee\langle b^d\rangle\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_1}}\ \vee$$
$$\overline{(\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_1}}\ *\qquad *_4$$
$$\overline{(\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q\wedge\langle b\rangle\langle b^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0,x_1}}\ \wedge$$
$$\overline{(\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0}}\ \text{clo}_{x_1}$$
$$\overline{(\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0}}\ \text{weak}$$
$$*_3$$

$$\overline{(p)^\varepsilon, (\neg p)^\varepsilon}\ \text{Ax1}$$
$$\overline{(p)^\varepsilon, (\neg p)^{x_0}}\ \text{exp}$$
$$\overline{(p)^{x_0}, (\neg p)^{x_0}}\ \text{exp}$$
$$\overline{(\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (p)^{x_0}, (\neg p)^{x_0}}\ \text{weak}\qquad *_3$$
$$\overline{(\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (p)^{x_0}, (\neg p\wedge\langle b^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0}}\ \wedge$$
$$\overline{(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^\varepsilon, (p)^{x_0}, (\neg p\wedge\langle b^\times\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))^\times\rangle q)^{x_0}}\ \text{weak}$$
$$*_1$$

$$*_1 \qquad *_2$$

$$(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (p \wedge \langle a^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0}, (\neg p \wedge \langle b^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0} \quad \wedge$$

$$(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (p \wedge \langle a^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0}, (\langle\neg p?\rangle\langle b^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0} \quad ?$$

$$(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (p \wedge \langle a^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0}, (\langle\neg p?; b^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0} \quad ;$$

$$(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle p?\rangle\langle a^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0}, (\langle\neg p?; b^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0} \quad ?$$

$$(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle p?; a^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0}, (\langle\neg p?; b^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0} \quad ;$$

$$(\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q \vee \langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle p?; a^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0}, (\langle\neg p?; b^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0} \quad \vee$$

$$(\langle a^{d^*}\sqcup b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle p?; a^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0}, (\langle\neg p?; b^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0} \quad \sqcup$$

$$(\neg q)^{\varepsilon}, (\langle a^{d^*}\sqcup b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle p?; a^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0}, (\langle\neg p?; b^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0} \quad \text{weak}$$

$$(\neg q \vee \langle a^{d^*}\sqcup b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle p?; a^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0}, (\langle\neg p?; b^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0} \quad \vee$$

$$(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle p?; a^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0}, (\langle\neg p?; b^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0} \quad *$$

$$(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle p?; a^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q \vee \langle\neg p?; b^{\times}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0} \quad \vee$$

$$(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle(p?; a^{\times})\sqcup(\neg p?; b^{\times})\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{x_0} \quad \sqcup$$

$$*_0$$

$$\overline{} \ \text{Ax1}$$

$$(\neg q)^{\varepsilon}, (q)^{\varepsilon}$$

$$(\neg q)^{\varepsilon}, (\langle a^{d^*}\sqcup b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (q)^{\varepsilon} \quad \text{weak}$$

$$(\neg q)^{\varepsilon}, (\langle a^{d^*}\sqcup b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (q)^{x_0} \quad \text{exp}$$

$$(\neg q \vee \langle a^{d^*}\sqcup b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (q)^{x_0} \quad \vee$$

$$(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (q)^{x_0} \quad * \qquad\qquad *_0$$

$$(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (q \wedge \langle(p?; a^{\times})\sqcup(\neg p?; b^{\times})\rangle\langle((p?; a^{\times})\sqcup(\neg p?; b^{\times}))^{\times}\rangle q)^{x_0} \quad \wedge$$

$$(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q)^{\varepsilon}, (\langle((p?; a^{\times})\sqcup(\neg p?; b^{\times}))^{\times}\rangle q)^{\varepsilon} \quad \text{clo}_{x_0}$$

$$(\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q \vee \langle((p?; a^{\times})\sqcup(\neg p?; b^{\times}))^{\times}\rangle q)^{\varepsilon} \quad \vee$$

In this tree there are two main branches each with applications of $\text{clo}_{\times}$. Note how even though they are on different branches, we use different names for the $\text{clo}_{\times}$ applications as we can not use the same name for different fixed point formulas.

3.2 The Proof System G

G is another sequent calculi but does not have the labels that CloG has and also does not have the $\text{clo}_{\times}$ rule. G instead has an induction rule that tracks the unfolding of greatest fixed point formulas. G has some other deep inference rules that allow manipulations on certain patterns in a formula regardless of where they appear. A propositional formula φ is proven in a valid G proof when the sequent at the root of the proof is the formula, and all of the branches of the proof result in an axiom. The rules for G are given in figure 3.3. In the figure, $\overline{\Phi}$ should be read as the negation of Φ. This negation function for complex propositional formulas in the language $\mathcal{L}_{\text{NF}}$, which also includes a dual function for complex games, is shown in Figure 3.2.

$$\begin{array}{lll} \overline{p} & := & \neg p \\ \overline{\neg p} & := & p \\ \overline{\varphi \vee \psi} & := & \overline{\varphi} \wedge \overline{\psi} \\ \overline{\varphi \wedge \psi} & := & \overline{\varphi} \vee \overline{\psi} \\ \overline{\langle \gamma \rangle \varphi} & := & \langle \widetilde{\gamma} \rangle \overline{\varphi} \end{array} \qquad \begin{array}{lll} \widetilde{g} & := & g^d \\ \widetilde{(g^d)} & := & g \\ \widetilde{\gamma\,;\delta} & := & \widetilde{\gamma}\,;\widetilde{\delta} \\ \widetilde{\gamma \sqcup \delta} & := & \widetilde{\gamma} \sqcap \widetilde{\delta} \\ \widetilde{\gamma \sqcap \delta} & := & \widetilde{\gamma} \sqcup \widetilde{\delta} \\ \widetilde{(\gamma^*)} & := & (\widetilde{\gamma})^\times \\ \widetilde{(\gamma^\times)} & := & (\widetilde{\gamma})^* \\ \widetilde{(\varphi?)} & := & \overline{\varphi}! \\ \widetilde{(\varphi!)} & := & \overline{\varphi}? \end{array}$$

Figure 3.2: Negation function from Defintion 6 of [3]

$$\frac{}{\Phi, \overline{\Phi}}\ \mathsf{Ax} \qquad \frac{\Phi}{\Phi, \varphi}\ \mathsf{weak} \qquad \frac{\varphi, \psi}{\langle g \rangle \varphi, \langle g^d \rangle \psi}\ \mathsf{mod}_m$$

$$\frac{\Phi, \varphi, \psi}{\Phi, \varphi \vee \psi}\ \vee \qquad \frac{\Phi, \varphi \qquad \Phi, \psi}{\Phi, \varphi \wedge \psi}\ \wedge \qquad \frac{\Phi, \psi(\gamma)}{\Phi, \psi(\chi!\,;\gamma)}\ \mathsf{Mon}^{\mathsf{g}}_{\mathsf{d}} \quad \frac{\Phi, \psi(\varphi)}{\Phi, \psi(\langle \chi! \rangle \varphi)}\ \mathsf{Mon}^{\mathsf{f}}_{\mathsf{d}} \quad \frac{\Phi, \psi(\langle \gamma \rangle \langle \delta \rangle \varphi)}{\Phi, \psi(\langle \gamma\,; \delta \rangle \varphi)}\ ;_d$$

$$\frac{\Phi, \varphi \vee \langle \gamma \rangle \langle \gamma^* \rangle \varphi}{\Phi, \langle \gamma^* \rangle \varphi}\ * \qquad \frac{\Phi, \langle \gamma \rangle \varphi \vee \langle \delta \rangle \varphi}{\Phi, \langle \gamma \sqcup \delta \rangle \varphi}\ \sqcup \qquad \frac{\Phi, \psi \wedge \varphi}{\Phi, \langle \psi? \rangle \varphi}\ ?$$

$$\frac{\Phi, \varphi \wedge \langle \gamma \rangle \langle \gamma^\times \rangle \varphi}{\Phi, \langle \gamma^\times \rangle \varphi}\ \times \qquad \frac{\Phi, \langle \gamma \rangle \varphi \wedge \langle \delta \rangle \varphi}{\Phi, \langle \gamma \sqcap \delta \rangle \varphi}\ \sqcap \qquad \frac{\Phi, \psi \vee \varphi}{\Phi, \langle \psi! \rangle \varphi}\ ! \qquad \frac{\Phi, \varphi \wedge \langle \gamma \rangle \langle (\overline{\Phi}!\,; \gamma)^\times \rangle \langle \overline{\Phi}! \rangle \varphi}{\Phi, \langle \gamma^\times \rangle \varphi}\ \mathsf{ind}_s$$

Figure 3.3: Axioms and Rules of G

The rules $\text{Mon}^{\mathsf{g}}_{\mathsf{d}}$, $\text{Mon}^{\mathsf{f}}_{\mathsf{d}}$ and $;_d$ are the deep inference rules of G. When they specify a formula of the pattern $\psi(\varphi)$ this specifies a formula ψ with an occurrence of the pattern φ within it. When a deep inference rule is applied the new pattern replaces φ in the same position. The ind_s rule, as described before, tracks the context of the unfolding of fixed point formulas. It does this by takingthe side formulas from the sequent and integrating their negation as demonic tests in the unfolded fixed point formula. We also have a different axiom rule that allows us to have not just an atomic proposition and its negation but a whole sequent and its negation at a leaf of the proof. This is also the only axiom in G. The rest of the rules are the same as the same named rules in CloG aside from the fact that the formulas in G do not have labels.

3.2.1 Example G Proof

Now let us look at an example proof in G. This proof is for the formula $\langle (a \sqcup a^d)^\times \rangle (p \vee \neg p)$:

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{Ax}
\UnaryInfC{$p, \neg p$}
\RightLabel{$\vee$}
\UnaryInfC{$p \vee \neg p$}
\AxiomC{}
\RightLabel{Ax}
\UnaryInfC{$p, \neg p$}
\RightLabel{$\vee$}
\UnaryInfC{$(p \vee \neg p)$}
\RightLabel{weak}
\UnaryInfC{$(p \vee \neg p), p \vee \neg p$}
\RightLabel{$\vee$}
\UnaryInfC{$(p \vee \neg p) \vee (p \vee \neg p)$}
\RightLabel{!}
\UnaryInfC{$\langle (p \vee \neg p)! \rangle (p \vee \neg p)$}
\AxiomC{}
\RightLabel{Ax}
\UnaryInfC{$p, \neg p$}
\RightLabel{$\vee$}
\UnaryInfC{$(p \vee \neg p)$}
\RightLabel{weak}
\UnaryInfC{$(p \vee \neg p), \langle (a \sqcup a^d) \rangle \langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p)$}
\RightLabel{$\vee$}
\UnaryInfC{$(p \vee \neg p) \vee \langle (a \sqcup a^d) \rangle \langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p)$}
\RightLabel{!}
\UnaryInfC{$\langle (p \vee \neg p)! \rangle \langle (a \sqcup a^d) \rangle \langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p)$}
\RightLabel{$;_d$}
\UnaryInfC{$\langle ((p \vee \neg p)!; (a \sqcup a^d)) \rangle \langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p)$}
\RightLabel{$\wedge$}
\BinaryInfC{$\langle (p \vee \neg p)! \rangle (p \vee \neg p) \wedge \langle ((p \vee \neg p)!; (a \sqcup a^d)) \rangle \langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p)$}
\RightLabel{$\times$}
\UnaryInfC{$\langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p)$}
\RightLabel{weak}
\UnaryInfC{$\langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p), \langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p)$}
\RightLabel{mod$_m$}
\UnaryInfC{$\langle a \rangle \langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p), \langle a^d \rangle \langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p)$}
\RightLabel{$\vee$}
\UnaryInfC{$\langle a \rangle \langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p) \vee \langle a^d \rangle \langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p)$}
\RightLabel{$\sqcup$}
\UnaryInfC{$\langle (a \sqcup a^d) \rangle \langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p)$}
\RightLabel{$\wedge$}
\BinaryInfC{$(p \vee \neg p) \wedge \langle (a \sqcup a^d) \rangle \langle ((p \vee \neg p)!; (a \sqcup a^d))^{\times} \rangle \langle (p \vee \neg p)! \rangle (p \vee \neg p)$}
\RightLabel{ind$_s$}
\UnaryInfC{$\langle (a \sqcup a^d)^{\times} \rangle (p \vee \neg p)$}
\end{prooftree}
$$

This particular example shows the case when we apply the induction rule with no side formulas. An empty sequent is an empty disjunction, which is equal to $\bot$. Since we do not have $\bot$ in the $\mathcal{L}_{\mathrm{NF}}$ language, we instead represent this using the law of excluded middle $p \vee \neg p$ where p is an arbitrary fixed atomic proposition. The negation of this, $p \vee \neg p$, is then inserted into the demonic tests.

3.3 CloG to G Transformation

Now that we have our two established proof systems we can look at the transformation from a CloG proof to a G proof. All transformations, functions, and lemmas defined in this section are as defined in section III.B in [3]. The first step of this is to be able to take a labelled formula in L_{NF} within a CloG sequent to it's equivalent non-labelled formula just in L_{NF}.

3.3.1 Bullet Translation

Before we get to this translation we first need some other notation. In Figure 3.4 there is a modality in which the modality is split wherever there is a ; operator. Then in Figure 3.5 φ is a sequence of formulas $\varphi_1, ..., \varphi_n$, the tests of which are sequentially concatenated in $\underleftarrow{\varphi!}$. This sequence of tests is also concatenated with another game γ in $\underleftarrow{\varphi! \cdot \gamma}$.

$$
\langle\!\langle \gamma \rangle\!\rangle \varphi \coloneqq \begin{cases} \langle\!\langle \gamma_1 \rangle\!\rangle \langle\!\langle \gamma_2 \rangle\!\rangle \varphi & \text{if } \gamma = \gamma_1 \, ; \gamma_2 \\ \langle \gamma \rangle \varphi & \text{otherwise} \end{cases}
$$

Figure 3.4: Split Modality Notation

$$
\varphi \coloneqq \varphi_1, ..., \varphi_n
$$

$$
\begin{aligned}
\underleftarrow{\varphi!} &\coloneqq \varphi_n! \,;(\ldots(\varphi_1!)\ldots) \\
\underleftarrow{\varphi! \cdot \gamma} &\coloneqq \varphi_n! \,;(\ldots(\varphi_1! \,; \gamma)\ldots).
\end{aligned}
$$

Figure 3.5: Game Concatenation Notation

The translation $^\bullet$ which we will call the bullet translation is defined in [3]. The input to the function is a propositional formula with a label $(\varphi)^{\mathsf{a}}$, and the output is a propositional formula ψ with demonic tests inserted into the fixed point formulas in φ that are named in a. If a is empty then $(\varphi)^{\mathsf{a}\bullet} = \psi$, otherwise, the patterns for inputs with non-empty labels are shown in Figure 3.6.

$$
\begin{aligned}
p^{\mathsf{a}\bullet} &\coloneqq p, & \beta(g, \mathsf{a}, \varphi) &\coloneqq g, \\
(\neg p)^{\mathsf{a}\bullet} &\coloneqq \neg p, & \beta(g^d, \mathsf{a}, \varphi) &\coloneqq g^d, \\
(\varphi \vee \psi)^{\mathsf{a}\bullet} &\coloneqq \varphi^{\mathsf{a}\bullet} \vee \psi^{\mathsf{a}\bullet}, & \beta(\psi?, \mathsf{a}, \varphi) &\coloneqq (\psi^{\mathsf{a}\bullet})?, \\
(\varphi \wedge \psi)^{\mathsf{a}\bullet} &\coloneqq \varphi^{\mathsf{a}\bullet} \wedge \psi^{\mathsf{a}\bullet}, & \beta(\psi!, \mathsf{a}, \varphi) &\coloneqq (\psi^{\mathsf{a}\bullet})!, \\
(\langle\gamma\rangle\varphi)^{\mathsf{a}\bullet} &\coloneqq \langle\!\langle \beta(\gamma, \mathsf{a}, \varphi) \rangle\!\rangle \varphi^{\mathsf{a}\bullet}, & \beta(\gamma^*, \mathsf{a}, \varphi) &\coloneqq \gamma^*,
\end{aligned}
$$

$$
\begin{aligned}
\beta(\gamma\,;\delta, \mathsf{a}, \varphi) &\coloneqq \beta(\gamma, \mathsf{a}, \langle\delta\rangle\varphi)\,;\beta(\delta, \mathsf{a}, \varphi), \\
\beta(\gamma \sqcup \delta, \mathsf{a}, \varphi) &\coloneqq \beta(\gamma, \mathsf{a}, \varphi) \sqcup \beta(\delta, \mathsf{a}, \varphi), \\
\beta(\gamma \sqcap \delta, \mathsf{a}, \varphi) &\coloneqq \beta(\gamma, \mathsf{a}, \varphi) \sqcap \beta(\delta, \mathsf{a}, \varphi).
\end{aligned}
$$

Figure 3.6: Bullet Translation

The last pattern is $\beta(\gamma^{\times}, \mathrm{a}, \varphi)$. In this case we take the sequence of names $x_1, ..., x_n$ in a of the fixed point formula $\langle\gamma^{\times}\rangle\varphi$. We assume that there is a formula χ_{x_i} assigned for every neame x in the CloG proof and, using these formulas, form the sequence $\chi = \chi_{x_1}, ..., \chi_{x_n}$.

We use the bullet translation to assign the formulas χ_{x} for each name x in the CloG proof. $\chi_{\mathsf{x}} = \overline{\Phi^\bullet}$ where Φ is the sequent of side formulas where the clo_{x} rule was applied. Here we assign these formulas in order of the clo_{x} application closest to the root of the tree, to the furthest.

3.3.2 Direct Transformations Of CloG Rules To G Rules

Now that we have a translation from sequents in CloG to sequents in G, we need a map from the rules in CloG to derivations in G. So we must take each rule application in CloG between two sequents Ψ and Φ, then we must find a derivation from $(\Psi)^\bullet$ to $(\Phi)^\bullet$ in G. This technique is as shown in Figure 3.7 where Π_1 is the G derivation.

$$
\frac{(A_0)^{\mathsf{a}_0}, ..., (A_n)^{\mathsf{a}_n}}{(B_0)^{\mathsf{b}_0}, ..., (B_n)^{\mathsf{b}_n}} \text{<CloG rule>}
$$

$$\Downarrow$$

$$(A_0)^{\mathsf{a}_0\bullet}, ..., (A_n)^{\mathsf{a}_n\bullet}$$

$$\Pi_1$$

$$(B_0)^{\mathsf{b}_0\bullet}, ..., (B_n)^{\mathsf{b}_n\bullet}$$

Figure 3.7: General transformation of a CloG rule application to a G derivation Π_1.

There is a direct map to a G rule for all CloG rules besides clo_{x}, exp, and the closure rule. In these cases we can bullet translate the sequents and replace the CloG rule with a G rule as shown in Figure 3.8. The direct map of CloG rules to G rules is given in Figure 3.9

$$\frac{(A_0)^{\mathsf{a}_0}, ..., (A_n)^{\mathsf{a}_n}}{(B_0)^{\mathsf{b}_0}, ..., (B_n)^{\mathsf{b}_n}} \text{<CloG rule>}$$

$$\Downarrow$$

$$\frac{(A_0)^{\mathsf{a}_0\bullet}, ..., (A_n)^{\mathsf{a}_n\bullet}}{(B_0)^{\mathsf{b}_0\bullet}, ..., (B_n)^{\mathsf{b}_n\bullet}} \text{<G rule>}$$

Figure 3.8: Direct transformation of a CloG rule application to a G rule application.

$$\begin{array}{rcl}
\text{Ax1} & \Rightarrow & \text{Ax} \\
\text{mod}_m & \Rightarrow & \text{mod}_m \\
\vee & \Rightarrow & \vee \\
\wedge & \Rightarrow & \wedge \\
\sqcup & \Rightarrow & \sqcup \\
\sqcap & \Rightarrow & \sqcap \\
\text{weak} & \Rightarrow & \text{weak} \\
; & \Rightarrow & ;_d \\
* & \Rightarrow & * \\
\times & \Rightarrow & \times \\
? & \Rightarrow & ? \\
! & \Rightarrow & !
\end{array}$$

Figure 3.9: Direct maps from CloG rules to G rules

3.3.3 Transformation Of A clo_{x} Rule Application

The first of the rules without a direct map is the clo_{x} rule. For this, we will have an application of the rule between the sequents of the patterns $\Phi, (\varphi \wedge \langle\gamma\rangle\langle\gamma^\times\rangle\varphi)^{\mathsf{ax}}$ and $\Phi, (\langle\gamma^\times\rangle\varphi)^{\mathsf{a}}$. Using the $^\bullet$ translation and Lemma 14 of [3] we have:

$$\begin{array}{rcl}
(\langle\gamma^\times\rangle\varphi)^{\mathsf{a}\bullet} & = & \langle(\underleftarrow{\chi! \cdot \gamma})^\times\rangle\langle\langle\chi!\rangle\rangle(\varphi^{\mathsf{b}\bullet}) \\
(\varphi \wedge \langle\gamma\rangle\langle\gamma^\times\rangle\varphi)^{\mathsf{ax}} & = & \varphi^{\mathsf{b}\bullet} \wedge \langle\langle\gamma\rangle\rangle\langle(\overline{\Phi^\bullet}!; \underleftarrow{\chi! \cdot \gamma})^\times\rangle\langle\langle\overline{\Phi^\bullet}!\rangle\rangle\langle\langle\chi!\rangle\rangle(\varphi^{\mathsf{b}\bullet})
\end{array}$$

Where the sequence χ contains the χ_{x} formulas for all of the names x in a for the fixed point formula $\langle\gamma^\times\rangle\varphi$. We transform the rule application like in Figure 3.7 using the pattern in Figure 3.10. The double horizontal lines in the derivation indicate that the rule can be applied multiple or no times to get the derivation. The number of $\text{Mon}^{\mathsf{f}}_{\mathsf{d}}$ and the number of $\text{Mon}^{\mathsf{g}}_{\mathsf{d}}$ applications are both the same as the number of formulas in the sequence χ. The number of $;_d$ applications is based on the number of sequential concatenations at the top level of the game γ.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\Phi^{\bullet}, \varphi^{\mathsf{b}\bullet} \wedge \langle\!\langle \gamma \rangle\!\rangle \langle (\overline{\Phi^{\bullet}!}; \underleftarrow{\chi! \cdot \gamma})^{\times} \rangle \langle \overline{\Phi^{\bullet}!} \rangle \langle\!\langle \underleftarrow{\chi!} \rangle\!\rangle \varphi^{\mathsf{b}\bullet}}{\Phi^{\bullet}, \varphi^{\mathsf{b}\bullet} \wedge \langle \gamma \rangle \langle (\overline{\Phi^{\bullet}!}; \underleftarrow{\chi! \cdot \gamma})^{\times} \rangle \langle \overline{\Phi^{\bullet}!} \rangle \langle\!\langle \underleftarrow{\chi!} \rangle\!\rangle \varphi^{\mathsf{b}\bullet}} ;_d}{\Phi^{\bullet}, \varphi^{\mathsf{b}\bullet} \wedge \langle \underleftarrow{\chi! \cdot \gamma} \rangle \langle (\overline{\Phi^{\bullet}!}; \underleftarrow{\chi! \cdot \gamma})^{\times} \rangle \langle \overline{\Phi^{\bullet}!} \rangle \langle\!\langle \underleftarrow{\chi!} \rangle\!\rangle \varphi^{\mathsf{b}\bullet}} \mathsf{Mon}^{\mathsf{g}}_{\mathsf{d}}}{\Phi^{\bullet}, \langle\!\langle \underleftarrow{\chi!} \rangle\!\rangle \varphi^{\mathsf{b}\bullet} \wedge \langle \underleftarrow{\chi! \cdot \gamma} \rangle \langle (\overline{\Phi^{\bullet}!}; \underleftarrow{\chi! \cdot \gamma})^{\times} \rangle \langle \overline{\Phi^{\bullet}!} \rangle \langle\!\langle \underleftarrow{\chi!} \rangle\!\rangle \varphi^{\mathsf{b}\bullet}} \mathsf{Mon}^{\mathsf{f}}_{\mathsf{d}}}{\Phi^{\bullet}, \langle (\underleftarrow{\chi! \cdot \gamma})^{\times} \rangle \langle\!\langle \underleftarrow{\chi!} \rangle\!\rangle \varphi^{\mathsf{b}\bullet},} \mathsf{ind}_s
$$

Figure 3.10: G derivation for a $\mathrm{clo}_{\mathsf{x}}$ application

3.3.4 Transformation Of A Closure Rule Application

Now we consider an application of the closure rule. Here we will have a sequent of the form $\Phi, \langle \gamma^{\times} \rangle \varphi^{\mathsf{ax}}$ and must find a G derivation from the bullet translation of this, to axioms as shown in figure 3.11. To do this we can use the G derivation pattern in Figure 3.12. Here, $\theta := \langle\!\langle \underleftarrow{\chi!} \rangle\!\rangle \varphi^{\mathsf{a}\bullet}$ and χ contains the χ_{x} formulas for all of the names x in a for the fixed point formula $\langle \gamma^{\times} \rangle \varphi$.

$$
[(A_0)^{\mathsf{a}_0}, ..., (\langle \gamma^{\times} \rangle \varphi)^{\mathsf{a}_n \mathsf{x}}]^{\mathsf{x}}
$$

$$
\Downarrow
$$

$$
\Pi_1
$$

$$
(A_0)^{\mathsf{a}_0 \bullet}, ..., (\langle \gamma^{\times} \rangle \varphi)^{\mathsf{a}_n \mathsf{x} \bullet}
$$

Figure 3.11: Transformation of a CloG closure to a G derivation

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Phi^{\bullet}, \chi_{\mathsf{x}}}{\Phi^{\bullet}, \chi_{\mathsf{x}}, \theta} \mathsf{weak}}{\Phi^{\bullet}, \chi_{\mathsf{x}} \vee \theta} \vee}{\Phi^{\bullet}, \langle \chi_{\mathsf{x}}! \rangle \theta} ! \quad \dfrac{\dfrac{\dfrac{\dfrac{\Phi^{\bullet}, \chi_{\mathsf{x}}}{\Phi^{\bullet}, \chi_{\mathsf{x}}, \langle \underleftarrow{\chi! \cdot \gamma} \rangle \langle (\chi_{\mathsf{x}}! \,; \underleftarrow{\chi! \cdot \gamma})^{\times} \rangle \langle \chi_{\mathsf{x}}! \rangle \theta} \mathsf{weak}}{\Phi^{\bullet}, \chi_{\mathsf{x}} \vee \langle \underleftarrow{\chi! \cdot \gamma} \rangle \langle (\chi_{\mathsf{x}}! \,; \underleftarrow{\chi! \cdot \gamma})^{\times} \rangle \langle \chi_{\mathsf{x}}! \rangle \theta} \vee}{\Phi^{\bullet}, \langle \chi_{\mathsf{x}}! \rangle \langle \underleftarrow{\chi! \cdot \gamma} \rangle \langle (\chi_{\mathsf{x}}! \,; \underleftarrow{\chi! \cdot \gamma})^{\times} \rangle \langle \chi_{\mathsf{x}}! \rangle \theta} !}{\Phi^{\bullet}, \langle \chi_{\mathsf{x}}! \,; \underleftarrow{\chi! \cdot \gamma} \rangle \langle (\chi_{\mathsf{x}}! \,; \underleftarrow{\chi! \cdot \gamma})^{\times} \rangle \langle \chi_{\mathsf{x}}! \rangle \theta} ;_d}{\Phi^{\bullet}, \langle \chi_{\mathsf{x}}! \rangle \theta \wedge \langle \chi_{\mathsf{x}}! \,; \underleftarrow{\chi! \cdot \gamma} \rangle \langle (\chi_{\mathsf{x}}! \,; \underleftarrow{\chi! \cdot \gamma})^{\times} \rangle \langle \chi_{\mathsf{x}}! \rangle \theta} \wedge}{\Phi^{\bullet}, \langle (\chi_{\mathsf{x}}! \,; \underleftarrow{\chi! \cdot \gamma})^{\times} \rangle \langle \chi_{\mathsf{x}}! \rangle \theta} \times
$$

Figure 3.12: G derivation to axioms from a CloG closure sequent

3.3.5 Transformation Of An exp Rule Application

When we have an application of the exp rule it will be between two sequents of the forms $\Phi, \varphi^{\mathsf{ab}}$ and $\Phi, \varphi^{\mathsf{axb}}$ and we will transform this as shown in Figure 3.7. Let $\langle \gamma^{\times} \rangle \varphi'$ be the fixed point formula named by x and call it θ. Take the sequence $\chi = \chi_{x_1}, .., \chi_{x_n}$ where $x_1, ..., x_n$ are the names for θ occurring ab. Let us also assign χ' in the same way with the names for θ that occur in axb. We know using the bullet translations, using the same deep matching notation as described in the rules of G, that we have:

$$
\varphi^{\mathsf{ab}\bullet} = \psi(\langle\!\langle (\underleftarrow{\chi! \cdot \gamma})^{\times} ; \underleftarrow{\chi!} \rangle\!\rangle \psi')
$$

$$
\varphi^{\mathsf{axb}\bullet} = \psi(\langle\!\langle (\underleftarrow{\chi'! \cdot \gamma})^{\times} ; \underleftarrow{\chi'!} \rangle\!\rangle \psi')
$$

Here the only difference in the formulas is the added formula χ_{x} in the sequence χ' that is not in χ. We can use a pattern of derivation as shown in Figure 3.13. We will have an application of $\text{Mon}_{\mathsf{d}}^{\mathsf{f}}$ and an application of $\text{Mon}_{\mathsf{d}}^{\mathsf{g}}$ for each occurrence of θ expanded by the bullet translation.

$$\cfrac{\cfrac{\psi(\langle\!\langle(\underleftarrow{\chi! \cdot \gamma})^{\times}; \underleftarrow{\chi!}\rangle\!\rangle\psi')}{\psi(\langle\!\langle(\underleftarrow{\chi! \cdot \gamma})^{\times}; \underleftarrow{\chi'!}\rangle\!\rangle\psi')}\,\text{Mon}_{\mathsf{d}}^{\mathsf{f}}}{\psi(\langle\!\langle(\underleftarrow{\chi'! \cdot \gamma})^{\times}; \underleftarrow{\chi'!}\rangle\!\rangle\psi')}\,\text{Mon}_{\mathsf{d}}^{\mathsf{g}}$$

Figure 3.13: G derivation for an exp rule application

3.3.6 Example Transformation

Now we can apply this transformation on a given CloG proof. For this example I will use the CloG proof from section 3.1.1. We can apply the transformation by starting from the root of the proof tree to the branches, first transforming a sequent with the bullet translation, then converting the above CloG rule to a G derivation, and then repeating for the next sequent. When the branches split we can also transform them separately one at a time. All parts of the proof still needed to be transformed will be highlighted in *blue*:

Right branch:

$$\cfrac{[(\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle(a^*;b)^{\times}\rangle p)^{x_0}]^{x_0}}{(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}\,\text{mod}_m$$

$$\cfrac{(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}{(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}, (\langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}\,\text{weak}$$

$$\cfrac{(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}, (\langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}{(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle b\rangle\langle(a^*;b)^{\times}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}\,\vee$$

$$\cfrac{(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle b\rangle\langle(a^*;b)^{\times}\rangle p \vee \langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}{(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle a^*\rangle\langle b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}\,*$$

$$\cfrac{(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle a^*\rangle\langle b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}{(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}\,;$$

$$\cfrac{(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}{(\neg p)^{\varepsilon}, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}\,\text{weak}$$

Left branch:

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

Combined:

$$\cfrac{(\neg p)^{\varepsilon}, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p)^{x_0} \qquad (\neg p)^{\varepsilon}, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}{(\neg p)^{\varepsilon}, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}\,\wedge$$

$$\cfrac{(\neg p)^{\varepsilon}, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}{(\neg p)^{\varepsilon}, (\langle a^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}\,\text{weak}$$

$$\cfrac{(\neg p)^{\varepsilon}, (\langle a^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}{(\neg p)^{\varepsilon}, (\langle a^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}\,\vee$$

$$\cfrac{(\neg p)^{\varepsilon}, (\langle a^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}{(\neg p)^{\varepsilon}, (\langle a^d \sqcup b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}\,\sqcup$$

$$\cfrac{(\neg p)^{\varepsilon}, (\langle a^d \sqcup b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}{(\neg p \vee \langle a^d \sqcup b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}\,\vee$$

$$\cfrac{(\neg p \vee \langle a^d \sqcup b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}{(\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}\,*$$

$$\cfrac{\cfrac{(\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p)^{x_0}}{(\langle(a^d \sqcup b^d)^*\rangle\neg p)^{\varepsilon}, (\langle(a^*;b)^{\times}\rangle p)^{\varepsilon}}\,\text{clo}_{x_0}}{(\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle(a^*;b)^{\times}\rangle p)^{\varepsilon}}\,\vee$$

First let us translate up to the clo_{x_0} application. The first two sequents are unchanged as they have empty labels, and the disjunction rule maps directly to the disjunction rule in G. Then to transform the clo_{x_0} application, we use ind_s to insert demonic tests of χ_{x_0}, which is calculated as $\overline{\langle(a^d \sqcup b^d)^*\rangle\neg p} = \langle(a \sqcap b)^{\times}\rangle p$, into the unfolded fixed point formula $p \wedge \langle a^*;b\rangle\langle(a^*;b)^{\times}\rangle p$. Then an application of $;_d$ is needed to split the concatenation in $a^*;b$.

$$[(\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle(a^*;b)^\times\rangle p)^{x_0}]^{x_0}$$

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

$$(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}, (\langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

$\vee$

$$(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b\rangle\langle(a^*;b)^\times\rangle p \vee \langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

$*$

$$(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

$;$

$$(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*;b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

weak

$$(\neg p)^\varepsilon, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*;b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

Left branch: Ax1

$$(\neg p)^\varepsilon, (p)^\varepsilon$$

exp

$$(\neg p)^\varepsilon, (p)^{x_0}$$

weak

$$(\neg p)^\varepsilon, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (p)^{x_0}$$

$\wedge$

$$(\neg p)^\varepsilon, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

weak

$$(\neg p)^\varepsilon, (\langle a^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

$\vee$

$$(\neg p)^\varepsilon, (\langle a^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

$\sqcup$

$$(\neg p)^\varepsilon, (\langle a^d \sqcup b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

$\vee$

$$(\neg p \vee \langle a^d \sqcup b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (p \wedge \langle a^*;b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

$*$

$$(\langle a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*\rangle\langle b\rangle\langle((\langle(a\sqcap b)^\times\rangle p!; a^*; b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p$$

$;_d$

$$(\langle a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle((\langle(a\sqcap b)^\times\rangle p!; a^*; b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p$$

ind_s

$$(\langle a^d \sqcup b^d)^*\rangle\neg p, \langle(a^*;b)^\times\rangle p$$

$\vee$

$$\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle(a^*;b)^\times\rangle p$$

Next we can translate up to the application of the conjunction. All rules up to and including it again have a direct map to a G rule.

$$[(\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle(a^*;b)^\times\rangle p)^{x_0}]^{x_0}$$

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

$$(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}, (\langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

$\vee$

$$(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b\rangle\langle(a^*;b)^\times\rangle p \vee \langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

$*$

$$(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

$;$

$$(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*;b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

weak

$$(\neg p)^\varepsilon, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*;b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}$$

Left branch: Ax1

$$(\neg p)^\varepsilon, (p)^\varepsilon$$

exp

$$(\neg p)^\varepsilon, (p)^{x_0}$$

weak

$$(\neg p)^\varepsilon, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (p)^{x_0}$$

$\wedge$

$$\neg p, \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle((\langle(a\sqcap b)^\times\rangle p!; a^*; b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p$$

weak

$$\neg p, (\langle a^d\rangle\langle a^d \sqcup b^d)^*\rangle\neg p, \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle((\langle(a\sqcap b)^\times\rangle p!; a^*; b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p$$

$\vee$

$$\neg p, (\langle a^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle b^d\rangle\langle a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle((\langle(a\sqcap b)^\times\rangle p!; a^*; b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p$$

$\sqcup$

$$\neg p, (\langle a^d \sqcup b^d\rangle\langle a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle((\langle(a\sqcap b)^\times\rangle p!; a^*; b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p$$

$\vee$

$$\neg p \vee \langle a^d \sqcup b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle((\langle(a\sqcap b)^\times\rangle p!; a^*; b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p$$

$*$

$$(\langle a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*\rangle\langle b\rangle\langle((\langle(a\sqcap b)^\times\rangle p!; a^*; b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p$$

$;_d$

$$(\langle a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle((\langle(a\sqcap b)^\times\rangle p!; a^*; b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p$$

ind_s

$$(\langle a^d \sqcup b^d)^*\rangle\neg p, \langle(a^*;b)^\times\rangle p$$

$\vee$

$$\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle(a^*;b)^\times\rangle p$$

Next we will translate the left branch. The only non-direct translation is the exp rule, however, the sentence being expanded contains no fixed point formula so there is no G derivation needed to apply the exp rule.

$$
\begin{array}{c}
[(\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle(a^*;b)^\times\rangle p)^{x_0}]^{x_0} \\ \hline
(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0} \quad \text{mod}_m \\ \hline
(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0}, (\langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0} \quad \text{weak} \\ \hline
(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle b\rangle\langle(a^*;b)^\times\rangle p \vee \langle a\rangle\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0} \quad \vee \\ \hline
(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*\rangle\langle b\rangle\langle(a^*;b)^\times\rangle p)^{x_0} \quad * \\ \hline
(\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*;b\rangle\langle(a^*;b)^\times\rangle p)^{x_0} \quad ; \\ \hline
(\neg p)^\varepsilon, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*;b\rangle\langle(a^*;b)^\times\rangle p)^{x_0} \quad \text{weak}
\end{array}
$$

$$
\begin{array}{c}
\neg p, p \quad \text{Ax1} \\ \hline
\neg p, \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \quad \text{weak}
\end{array}
$$

$$
\begin{array}{c}
\neg p, \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \qquad (\neg p)^\varepsilon, (\langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p)^\varepsilon, (\langle a^*;b\rangle\langle(a^*;b)^\times\rangle p)^{x_0} \\ \hline
\neg p, \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad \wedge \\ \hline
\neg p, (\langle a^d\rangle\langle a^d \sqcup b^d)^*\rangle\neg p, \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad \text{weak} \\ \hline
\neg p, (\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, p \wedge \langle a^*;b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad \vee \\ \hline
\neg p, (\langle a^d \sqcup b^d\rangle\langle a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad \sqcup \\ \hline
\neg p \vee \langle a^d \sqcup b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad \vee \\ \hline
(\langle a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*\rangle\langle b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad * \\ \hline
(\langle a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad ;_d \\ \hline
(\langle a^d \sqcup b^d)^*\rangle\neg p, \langle(a^*;b)^\times\rangle p \quad \text{ind}_s \\ \hline
\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle(a^*;b)^\times\rangle p \quad \vee
\end{array}
$$

Now moving to the right branch, a thing to note is that, when earlier transforming the clo_{x_0} rule to a G derivation, we split the $a^*;b$ game already. This means that the application of the ; rule on this branch does not need to be applied and can be removed in the G proof. This is something to take into account when transforming the rules with a direct map to a G rule. After the application of the $*$ rule on this branch, the bullet translation causes the formulas to get much longer so the proof needs to be split up. A $\ast$ symbol is used to link one proof fragment to the other.

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

$$
\begin{array}{c}
\neg p, p \quad \text{Ax1} \\ \hline
\neg p, \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \quad \text{weak} \qquad \ast_0
\end{array}
$$

$$
\begin{array}{c}
\neg p, \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \qquad \ast_0 \\ \hline
\neg p, \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*\rangle\langle b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad \wedge \\ \hline
\neg p, (\langle a^d\rangle\langle a^d \sqcup b^d)^*\rangle\neg p, \langle b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*\rangle\langle b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad \text{weak} \\ \hline
\neg p, (\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, p \wedge \langle a^*\rangle\langle b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad \vee \\ \hline
\neg p, (\langle a^d \sqcup b^d\rangle\langle a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*\rangle\langle b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad \sqcup \\ \hline
\neg p \vee \langle a^d \sqcup b^d\rangle\langle(a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*\rangle\langle b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad \vee \\ \hline
(\langle a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*\rangle\langle b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad * \\ \hline
(\langle a^d \sqcup b^d)^*\rangle\neg p, p \wedge \langle a^*;b\rangle\langle(\langle(a\sqcap b)^\times\rangle p!; a^*;b)^\times\rangle\langle\langle(a\sqcap b)^\times\rangle p!\rangle p \quad ;_d \\ \hline
(\langle a^d \sqcup b^d)^*\rangle\neg p, \langle(a^*;b)^\times\rangle p \quad \text{ind}_s \\ \hline
\langle(a^d \sqcup b^d)^*\rangle\neg p \vee \langle(a^*;b)^\times\rangle p \quad \vee
\end{array}
$$

Lastly the closure can be transformed to a G derivation according to the pattern in Figure 3.12.

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{Ax}
\UnaryInfC{$\langle (a^d \sqcup b^d)^* \rangle \neg p, \langle (a \sqcap b)^\times \rangle p$}
\RightLabel{weak}
\UnaryInfC{$\langle (a^d \sqcup b^d)^* \rangle \neg p, \langle (a \sqcap b)^\times \rangle p, p$}
\RightLabel{$\vee$}
\UnaryInfC{$\langle (a^d \sqcup b^d)^* \rangle \neg p, \langle (a \sqcap b)^\times \rangle p \vee p$}
\RightLabel{!}
\UnaryInfC{$\langle (a^d \sqcup b^d)^* \rangle \neg p, \langle \langle (a \sqcap b)^\times \rangle p! \rangle p$}
\AxiomC{}
\RightLabel{Ax}
\UnaryInfC{$\langle (a^d \sqcup b^d)^* \rangle \neg p, \langle (a \sqcap b)^\times \rangle p$}
\RightLabel{weak}
\UnaryInfC{$\langle (a^d \sqcup b^d)^* \rangle \neg p, \langle (a \sqcap b)^\times \rangle p, \langle a^*; b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{$\vee$}
\UnaryInfC{$\langle (a^d \sqcup b^d)^* \rangle \neg p, \langle (a \sqcap b)^\times \rangle p \vee \langle a^*; b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{!}
\UnaryInfC{$\langle (a^d \sqcup b^d)^* \rangle \neg p, \langle \langle (a \sqcap b)^\times \rangle p! \rangle \langle a^*; b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{$;_d$}
\UnaryInfC{$\langle (a^d \sqcup b^d)^* \rangle \neg p, \langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b) \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{$\wedge$}
\BinaryInfC{$\langle (a^d \sqcup b^d)^* \rangle \neg p, \langle \langle (a \sqcap b)^\times \rangle p! \rangle p \wedge \langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b) \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{$\times$}
\UnaryInfC{$\langle (a^d \sqcup b^d)^* \rangle \neg p, \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{mod_m}
\UnaryInfC{$\langle b^d \rangle \langle (a^d \sqcup b^d)^* \neg p, \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p \rangle$}
\RightLabel{weak}
\UnaryInfC{$\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p, \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p, \langle a \rangle \langle a^* \rangle \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{$\vee$}
\UnaryInfC{$\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p, \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p \vee \langle a \rangle \langle a^* \rangle \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{$*$}
\UnaryInfC{$\langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p, \langle a^* \rangle \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{weak}
\UnaryInfC{$\neg p, \langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p, \langle a^* \rangle \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; (a^*; b))^\times \rangle \langle \langle a \sqcap b)^\times \rangle p! \rangle p$}
\noLine
\UnaryInfC{$*_0$}
\end{prooftree}
$$

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{Ax1}
\UnaryInfC{$\neg p, p$}
\RightLabel{weak}
\UnaryInfC{$\neg p, \langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p, p$}
\AxiomC{$*_0$}
\RightLabel{$\wedge$}
\BinaryInfC{$\neg p, \langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p, p \wedge \langle a^* \rangle \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; a^*; b)^\times \rangle \langle \langle (a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{weak}
\UnaryInfC{$\neg p, (\langle a^d \rangle \langle a^d \sqcup b^d)^* \rangle \neg p, \langle b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p, p \wedge \langle a^* \rangle \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; a^*; b)^\times \rangle \langle \langle (a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{$\vee$}
\UnaryInfC{$\neg p, (\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, p \wedge \langle a^* \rangle \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; a^*; b)^\times \rangle \langle \langle (a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{$\sqcup$}
\UnaryInfC{$\neg p, (\langle a^d \sqcup b^d \rangle \langle a^d \sqcup b^d)^* \rangle \neg p, p \wedge \langle a^* \rangle \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; a^*; b)^\times \rangle \langle \langle (a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{$\vee$}
\UnaryInfC{$\neg p \vee \langle a^d \sqcup b^d \rangle \langle (a^d \sqcup b^d)^* \rangle \neg p, p \wedge \langle a^* \rangle \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; a^*; b)^\times \rangle \langle \langle (a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{$*$}
\UnaryInfC{$(\langle a^d \sqcup b^d)^* \rangle \neg p, p \wedge \langle a^* \rangle \langle b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; a^*; b)^\times \rangle \langle \langle (a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{$;_d$}
\UnaryInfC{$(\langle a^d \sqcup b^d)^* \rangle \neg p, p \wedge \langle a^*; b \rangle \langle (\langle \langle (a \sqcap b)^\times \rangle p!; a^*; b)^\times \rangle \langle \langle (a \sqcap b)^\times \rangle p! \rangle p$}
\RightLabel{ind_s}
\UnaryInfC{$(\langle a^d \sqcup b^d)^* \rangle \neg p, \langle (a^*; b)^\times \rangle p$}
\RightLabel{$\vee$}
\UnaryInfC{$\langle (a^d \sqcup b^d)^* \rangle \neg p \vee \langle (a^*; b)^\times \rangle p$}
\end{prooftree}
$$

4 Existing Literature on Proof Transformation Implementation

Automated Theorem Proving (ATP) is a wide area within computing science. While there is no literature currently for game logic implementations, there are plenty of examples of logical proof implementations [4, 5, 6]. These implementations range from simple proof visualisation to full proof search tools. Some of these also utilise transformations similar to the one I will implement.
The most prominent form of proof transformation in logic is cut-elimination. This cut-elimination, introduced by Gentzen [7], is a transformation to derive an analytic proof from a synthetic proof. This means removing any lemma rules and instead, all statements in the proof are sub-formulas of the root formula. One method of cut-elimination, designed for Gentzen's LK-proofs, is CERES which is based on resolution [8]. An implementation of the CERES system is presented in this paper [9]. For this system the proofs are represented in the well known data representation language XML. An example of this representation is shown in Figure 4.1.

```
<proof symbol="the-proof" calculus="LK">
 <rule symbol="c:r" type="contrr" param="2">
  <sequent>...</sequent>
  <rule symbol="cut" type="cut">
   <sequent>...</sequent>
   <rule symbol="\pi:r" type="permr" param="(1 2)">
    <sequent>...</sequent>
    <rule symbol="cut" type="cut">
     <sequent>...</sequent>
     <prooflink symbol="\tau"/>
     <prooflink symbol="\epsilon(0)"/>
    </rule>
   </rule>
   <prooflink symbol="\epsilon(1)"/>
  </rule>
 </rule>
</proof>
```

Figure 4.1: XML proof representation

5 Requirements for the Transformation Tool

In order to keep all the aims and priorities of the project clear I have set out a list of requirements for the transformation tool. I have chosen to prioritise these requirements into must have, should have and could have. These priorities help with planning the project as I can make sure I tackle the highest priority task first and then add additional requirements later. I also have a section of non-functional requirements that will not be features of the tool but should be kept in mind during the project to produce a high quality product. I can later refer back to these requirements when testing to check I have covered all of them and I can also go over the requirements when I evaluate the results of the project.

5.1 Must Have

This is the base functionality of the tool for it be at all usable for what it is intended.

R-1.1: The tool must have an abstract digital representation of the language $\mathcal{L}_{\mathrm{NF}}$.

R-1.2: The tool must have an abstract digital representation of a CloG proof tree.

R-1.3: The tool must have an abstract digital representation of a G proof tree.

R-1.4: The tool must be able to transform a CloG proof tree to a G proof tree deriving the same sequent.

5.2 Should Have

This functionality is very important for the usability of the tool.

R-2.1: A concrete syntax should be defined in which a user can define a CloG proof tree.

R-2.2: The tool should be able to parse and transform the concrete syntax for a CloG proof tree.

R-2.3: The tool should be able to output a G proof tree styled in LaTeX.

5.3 Could Have

This functionality would just be additional to the tool in order to make it even more usable, portable, and extendable.

R-3.1 The tool could have a main function that can be called in order to perform all the main steps of the tool in one. These steps being:

1. R-2.2

2. R-1.4
3. R-2.3

R-3.2 The tool could be compiled into a portable executable that can read a CloG proof tree and output a G proof tree

R-3.3 The tool could have an abstract digital representation of the full syntax of game logic

5.4 Non-Functional Requirements

As mentioned before these requirements do not represent features of the tool but instead are standards that the tool's development should keep to. They will also have the priority of the "should have" requirements.

R-4.1 The tool should have modular code so that functions can be re-used for other game logic rascal tools more easily.

R-4.2 The tool's code should not be unnecessarily inefficient in a way that it would struggle to transform large proofs quickly.

R-4.3 The tool's code should have detailed comments in order to make it easy to understand, develop, and re-use.

6 Implementation

Now that the context is established, the implementation of the tool can be discussed. Before this discussion, a list of technologies used to develop the tool and their details will be given.

6.1 Development Technology

- **Rascal**
  The language chosen to implement the tool was Rascal (see [10]). Rascal is a java based language built for meta-programming. While the intended use of Rascal is for developing software languages, applying it to proof trees served a similar purpose. Rascal allows for easy definition of concrete and abstract syntax, parsing, functional and imperative style programming, pattern matching, deep matching, tree traversal and set evaluation. All of which were well utilised for this case of defining formulas, proof tree structures and transformations.

- **Eclipse IDE**
  Rascal can be compiled on the command line for testing, however, using the Eclipse IDE plugin for Rascal development is much more efficient. This plugin is recommended on Rascal's website, `https://www.rascal-mpl.org/`, along with instructions on how to install and use it. The plugin automatically builds the project as it is developed, and gives real time in depth static analysis and error checking while coding. This saves a significant amount of time in bug fixing after coding, as almost all syntax and semantic errors are fixed before running. Only logical error fixing is required when testing.

6.2 Structure

Within the Rascal project, multiple directories are used to organise the different files for the tool. Starting with the src folder for the Rascal code, the input folder for the input CloG files, and the output folder for the output LaTeX files. The Rascal code is then split up into multiple modules based on their functionality:

- GLASTs - defining the abstract representations of CloG, G, and game logic.
- CloGSyntax - defining the concrete input syntax for CloG.

- CST2AST_CloG - contains the functions for converting a parsed syntax tree for the CloG input to the abstract representation of CloG.
- LaTeXOutput - contains the functions that take an abstract representation of CloG, G, or game logic and output code for a LaTeX visual representation.
- BulletTranslate - contains the functions required for χ_{x} name assignment and the bullet translation of a labelled formula to its corresponding non-labelled formula.
- CloG2G - contains the functions that convert a valid CloG proof to a valid G proof for the same sequent.
- GLTool - main module containing the function to handle input, conversion, and LaTeX output for the transformation from CloG to G. Also contains the function to activate the eclipse IDE support for CloG input files.

6.3 Abstract Syntax

In this project, the system used in order to represent game logic expressions, CloG proofs and G proofs is Rascal's for declaring Algebraic data types. Once declared, these can be declared as a variable and used like any other type such as integers and booleans. These types are defined by a list of possible forms they can take, much like grammar definitions, and can contain references to other data types. They can also be defined recursively, referring to another object of the same type.

The definitions start from the atomic values, these being the atomic propositions, atomic games, and name identifiers. These can take two possible forms, either just referencing a string for their name or referencing a string name along with a subscript integer. Above atoms are the game logic formulas, which are the same for both proof systems, using all of the operators and abbreviations defined for game logic in Section 2.3. Game logic has two types, propositions and games, which are defined mutually recursively. In addition to the defined syntax, annotated versions of demonic iteration and the angelic modality that also contain an integer are added. This integer annotation is used later when transforming the exp rule from CloG to a G derivation.

Moving up to the proof systems, CloG formulas are labelled within sequents. Therefore, an extra type for a CloG *term* which contains a formula and a list of name identifiers for its label is added. In this CloG representation, all sequents are list of CloG terms. In G, they are all list of game logic propositions. Additionally, types for each proof system's set of rules have been added. Most of these rules are just named and do not refer to any other data, except for the clo_{x} rule which refers to the name identifier x that is introduced in its application. Both proof systems are then structured in the same way, with the proof split into sequents paired with the rules that are applied above them, referring to each other from the root of the proof to the branches. This forms a recursive chain of sequents and rules from the root to the branches. These nodes of the proof are referred to as the *proof parts*, which can take a few different forms for each proof system. By the recursive structure of the proofs, any proof part can be taken as a root of a sub-proof, which allows for easy recursive operations on the entire proof tree.

A proof part in my CloG representation can be one of four forms. It can be a leaf from an axiom, a closure application, a unary inference, or a binary inference. The leaf is atomic and does not refer to any other data. A closure application contains the sequent and the name for the fixed point formula that was previously unfolded in the same context. A unary inference contains the sequent, the CloG rule applied above it, and a reference to the next proof part above it. Finally, a binary inference contains the sequent and two references to the two proof parts above it. The binary inference does not contain a rule as there is only the $\wedge$ binary operator in CloG. The proof parts for the G representation are the same apart having no closure rule form. G proof parts also contain G sequents and G rules rather than the CloG representations.

6.4 Implementing The CloG to G Transformation

6.4.1 Implementing The Bullet Translation

The first implementation for the transformation is the bullet translation of labelled sequents from the CloG proofs to non-labelled sequents for a G proof, the definition of which is described in Section 3.3.1. This bullet translation has a few dependencies defined. The first of these being the $\langle\langle\gamma\rangle\rangle\varphi$ notation that splits any concatenations on the top level of a modality into multiple modalities as shown in Figure 3.4. It is not certain how these concatenations are bracketed and a chain of modalities is not represented as a list but instead are linked recursively. Therefore, there is a function that first generates a list of the sequentially concatenated games in the top level of γ, then a reduction statement builds the chain of modalities recursively. The next notation is $\underleftarrow{\varphi!}$ and $\underleftarrow{\varphi! \cdot \gamma}$ for a sequence of propositions $\varphi = \varphi_0, ..., \varphi_n$ that represents the concatenation of demonic tests of all the propositions in the sequence φ and the same but additionally concatenated with a game γ respectively, see Figure 3.5. A reduction statement concatenated the tests from the sequence φ. The negation of a propositional formula $\overline{\varphi}$ in $\mathcal{L}_{\mathrm{NF}}$ is also implemented, and uses pattern matching for the negation or dual of each operator. This uses the negation function shown in Figure 3.2.

Bullet translation also requires a mapping of each name x to a formula χ_{x}. It was decided that it was easiest to first resolve, from a whole CloG proof, a relation of each name x to data on the sequent at the $\mathrm{clo}_{\mathsf{x}}$ application. By storing more than just the χ_{x} formula in the relation, it can be used by both the bullet translation and the translation itself. The relation uses the alias CloNames and relates each name x to a tuple with the alias CloSeq. This CloSeq tuple contains: the list of bullet translated side formulas $\Phi^{\bullet}$ from the $\mathrm{clo}_{\mathsf{x}}$ application, the assigned χ_{x} formula, and the fixed point formula unfolded by the $\mathrm{clo}_{\mathsf{x}}$ application. A rascal visit statement is used to search a proof tree, from the root to the leaves, and add a relation to the CloNames variable for each $\mathrm{clo}_{\mathsf{x}}$ application that appears in the proof. The problem here is that in order to calculate the χ_{x}, the bullet translation is needed, which may require some other χ_{x} formulas. By constructing the relation from the root to the leaves, it is ensured that at each clox that is encountered, the bullet translation will be well defined using only the names of the clox applications closer to the root. This is a property of bullet translation also discussed in Section 3.3.1. This means that the current state of the relation can be used for the bullet translation during the search for $\mathrm{clo}_{\mathsf{x}}$ rules, and will be sufficient to calculate the next χ_{x} formula.

The bullet translation function itself uses pattern matching on the main operator in the formula φ that is input. The label of the formula a and the CloNames relation ns is also input. The $\beta(\gamma, \mathsf{a}, \varphi)$ function is implemented with pattern matching on the main operator in the game γ. This also has the additional inputs of the label a and CloNames relation ns, plus the propositional formula φ. In the case of the demonic iteration, the names corresponding to the fixed point formula $\langle\gamma\rangle\psi$ in the label are identified, then resolved in the CloNames relation to get the list of χ_{x} formulas, then the notation functions defined before are called to insert the demonic tests into the fixed point formula. Figure 3.6 can also be referred to in order to understand this function, as it uses the same input-output patterns and recursive function calls as shown in the figure.

6.4.2 Mapping Rules to Derivations

I defined a recursive pattern matching function on the CloG parts that translates the proofs from the root to the leaves. The base cases of this translation is at either a leaf (which remains a leaf but instead the G representation), an axiom (which returns a G axiom with the bullet translated sequent, see Section 3.3.2), or the closure rule. In the case of the closure rule, the active formula is identified by its label containing the name x that is also in the closure proof part type. Then, using Rascal's pattern matching, the bullet translated active formula is split into its relevant parts as specified in the transformation definition in Section 3.3.4. Using these parts, the exact pattern of G derivation from the sequent to axioms in Figure 3.11 is simply output with the arbitrary formulas inserted. There is one alternate case, when the $\mathrm{clo}_{\mathsf{x}}$ application has no side formulas and so the χ_{x} formula is $\top$. In this case, the $\top$ is represented as $p \vee \neg p$ where p is an arbitrary fixed propositional atom. One additional $\vee$ rule application is inserted to split the $p \vee \neg p$ formula before the axioms for the proof to then be valid.

For any unary inference cases where the rule has a direct map as described in Section 3.3.2, a function to map from each of these CloG rules to the G rule is defined as shown in Figure 3.9. Additionally, there is a check for if the sequents before and after the rule are already equal, as happened in one ; rule application case from the example translation in Section 3.3.6. If they are equal then the rule is skipped and the recursive translation function is called on the rest of the tree.

In the case of a clo_{x} application, we refer to the translation in Section 3.3.3. The active formula is identified by the difference between the sequents before and after the rule application. The G derivation pattern transformed from a clo_{x}, as in Figure 3.12, contains some rule applications that may be applied zero or more times. In order to handle this, separate recursive functions are defined for each rule in the derivation patter that can be applied within the transformation pattern. Each of these functions will recursively iterate to apply the rules, then at the base case when the rule does not need to be applied again the next rule function is called. The functions for the $\text{Mon}^{\mathsf{f}}_{\mathsf{d}}$ and $\text{Mon}^{\mathsf{g}}_{\mathsf{d}}$ rules apply the number of rules based on the number of formulas in the χ sequence. Then the $;_d$ function applies the rules based on the number of sequentially concatenated games in the top level of γ. These concatenated games are identified using the same function described in the first paragraph of Section 6.4.1. Then at the end of the last function the recursive transformation function is called on the rest of the proof tree.

The last case to be covered is the exp rule, the transformed of which is described in Section 3.3.5. This was the most difficult to implement due to Rascal's functionality for altering parts of an algebraic data type. Rascal's visit statements allow for the replacement of patterns within an algebraic data type, however, this replaces all instances of that pattern at once. The pattern in Figure 3.13 shows that a $\text{Mon}^{\mathsf{g}}_{\mathsf{d}}$ and a $\text{Mon}^{\mathsf{f}}_{\mathsf{d}}$ rule must be applied multiple times for each occurrence of θ expanded by the bullet translation, of which there may be multiple. The sequents at each step of these rule applications must be formed by editing these occurrences one at a time. This problem is solved by using some annotated versions of the demonic iteration and modality operators for game logic mentioned in Section 6.3, and an altered version of bullet translation. The altered bullet translation adds a 0 annotation wherever an occurrence of the fixed point formula θ is expanded. Then, using visit, an arbitrary order of integers is given to these annotated occurrences. After this, a loop over visit statements is used to apply a $\text{Mon}^{\mathsf{g}}_{\mathsf{d}}$ and a $\text{Mon}^{\mathsf{f}}_{\mathsf{d}}$ one by one for each occurrence of an annotation, as they are differentiated by their integer annotations rather than being identical patterns.

6.5 Interaction With the Tool

With the base functionality for the transformation defined, a working tool that meets the minimum aims of this project is obtained. However, the tool with only the implementations discussed up to this point is not easy to use, as declaring the algebraic types for the proof systems and game logic manually is tedious and prone to errors. The output is also not easily readable. The following sections describe the implementation of the parts of the tool intended for better usability of the tool.

6.5.1 CloG Input Syntax

Rascal's integrated system for syntax definition and parsing was used for the CloG input syntax. The definition of this syntax is similar to defining an algebraic data type, however, the forms are strings of ASCII characters or other non-terminal syntax patterns. Patterns can also be included with a * in order to allow that pattern to appear a repeated number of times, zero or more. The syntax definition has a starting pattern and then is parsed left to right. It is also implemented so that there is no ambiguity at any point for a left to right parser on which form a syntax pattern is taking, which is a detail that must be ensured by the coder and is not automatically handled.

A CloG input file must always start with "CloG" and then the proof is surrounded with braces. Beginning with "CloG" is not functionally necessary but this makes it clear what the file is when viewing. The proof itself is split into a list of proof parts, the same as the algebraic data type. Each of these parts is ended with a semi-colon. In the case of a unary inference or the closure rule, there is a sequent followed by the name

of the rule being applied. When using the clo_x or the closure rule, the additional corresponding name must also be included. In the case of a binary inference, the corresponding two branches are contained by two more pairs of braces nested within the surrounding ones. A sequent is surrounded by square brackets and containing a list of game logic formulas paired with labels. Labels are also surrounded by square brackets and contain a list of names.

Rascal also has a binding priority system for expressions, which was used both for propositions and games. The unary operators have priority. The negation and dual operators are only usable with an atomic proposition or game. In terms of the syntax, the identifiers for atoms and names are same as they have the same form of a string and an optional subscript number. The identifiers have no need to be distinguished within the syntax as their type can be identified by their location in the proof. An example CloG input proof is show in Figure 6.1.

```
CloG {
        [(<(a^d || b^d)*>~p | <(a*;b)^x>p)^[]] or;
        [(<(a^d || b^d)*>~p)^[] (<(a*;b)^x>p)^[]] clo_ x_0;
        [(<(a^d || b^d)*>~p)^[] (p & <a*;b><(a*;b)^x>p)^[x_0]] iter;
        [(~p | <a^d || b^d><(a^d || b^d)*>~p)^[] (p & <a*;b><(a*;b)^x>p)^[x_0]] or;
        [~p^[] <a^d || b^d><(a^d || b^d)*>~p^[] (p & <a*;b><(a*;b)^x>p)^[x_0]] choice;
        [~p^[] (<a^d><(a^d || b^d)*>~p | <b^d><(a^d || b^d)*>~p)^[] (p & <a*;b><(a*;b)^x>p)^[x_0]] or;
        [~p^[] <a^d><(a^d || b^d)*>~p^[] <b^d><(a^d || b^d)*>~p^[] (p & <a*;b><(a*;b)^x>p)^[x_0]] weak;
        [~p^[] <b^d><(a^d || b^d)*>~p^[] (p & <a*;b><(a*;b)^x>p)^[x_0]] and;
        {
                [~p^[] <b^d><(a^d || b^d)*>~p^[] (p)^[x_0]] weak;
                [~p^[] (p)^[x_0]] exp;
                [~p^[] (p)^[]] Ax1;
                leaf;
        }
        {
                [~p^[] <b^d><(a^d || b^d)*>~p^[] <a*;b><(a*;b)^x>p^[x_0]] weak;
                [<b^d><(a^d || b^d)*>~p^[] <a*;b><(a*;b)^x>p^[x_0]] concat;
                [<b^d><(a^d || b^d)*>~p^[] <a*><b><(a*;b)^x>p^[x_0]] iter;
                [<b^d><(a^d || b^d)*>~p^[] (<b><(a*;b)^x>p | <a><a*><b><(a*;b)^x>p)^[x_0]] or;
                [<b^d><(a^d || b^d)*>~p^[] <b><(a*;b)^x>p^[x_0] <a><a*><b><(a*;b)^x>p^[x_0]] weak;
                [<b^d><(a^d || b^d)*>~p^[] <b><(a*;b)^x>p^[x_0]] modm;
                [<(a^d || b^d)*>~p^[] <(a*;b)^x>p^[x_0]] closure^ x_0;
                leaf;
        }
}
```

Figure 6.1: CloG input file of the proof in Section 3.1.1

A function to transform this syntax to the algebraic data type is also defined. For this, different functions for each type of syntax pattern are used. Within each function, a switch statement recognises which form the syntax takes. Rascal reductions statements are utilised in order to iterate over syntax lists and build data types from proof parts, sequents, and labels. Identifiers, while having no distinction between the syntax versions, are cast to the correct algebraic type based on where they appear within the proof structure.

With the Rascal Eclipse plugin, it was possible to create a function in the GLTool module that registers this CloG syntax to be used with files that have the extension *.clog*. Running the IDE() function adds key word highlighting and real time syntax checking when editing a .clog file within Eclipse.

6.5.2 LaTeX output

Functions are defined for the output of a Tex file visualising both G proofs and CloG proofs. For each of these, a string is built and then written to a Tex file. The first functions defined provides the wrapper for the Tex file. In this wrapper, the document is defined as an article, and the bussproofs.sty style package is imported to be used to present the proof trees. From the wrapper function, a function is called, given either

a CloG or G proof tree type as input, to built the proof tree according to the style package.

This proof tree function works recursively in the same way as the conversion from the syntax to algebraic data types The difference is that instead of switch statements, pattern matching is used to convert to different strings based on the type of the input. Within the string building, for loops are used to generate the list of proof parts as inferences with labels. Additionally, reduction statements are used to convert the sequent lists and name lists. A function is also added that after calling the main LaTeX output function, writes the string to a Tex file in the output folder.

6.5.3 Main Function

The main function in the GLTool module simply calls the functions from the other modules but makes it easier to handle for the user. The function takes a string as input with the name of the CloG file that the user wants to transform. The function reads the .clog file with that name from the input folder, converts the syntax to the algebraic type, transforms the CloG proof to a G proof, converts the algebraic type to the LaTeX representation, then creates a Tex file of the same name as the input file in the output folder. This function is split into multiple smaller functions that were used for manual testing in the terminal during development.

7 Testing

There are two main categories to the testing procedure, automated testing and manual testing. For automated testing, Rascal. test keyword was used. This keyword can be applied to any function returning a boolean. When running the command :test in the rascal terminal, any test functions in any imported modules will be run and any false results are reported. In the case of these functions having input, Rascal will also generate some randomised input.

7.1 Automated Testing

The modules for concrete syntax and algebraic data type definitions can not be automatically tested as they do not contain functions. However, the use of these will be tested when testing other functions that use these types.

Most functions are tested with several test cases in order to assert that they are functioning as intended. Some functions, however, have less test cases due to the fact that they are called and tested when other functions are directly tested. One example of this is the functions for conversion of certain syntactic indentifiers to the algebraic CloG data types for names and atoms, see Section 6.3. These functions are used in test cases for the other conversion functions and have limited functionality, therefore they required less test cases. The syntax type inputs for most of the syntax converstion tests are directly hardcoded. The exception to this is for the main conversion function, where strings containing a full proof are hardcoded and then parsed for input. This also tests the parsing functionality for CloG input.

Next the tests for the BulletTranslate module, see Section 6.4.1. Testing the notation functions and negation functions only required a few simple test cases to make sure they are functioning correctly. The function for resolving the clo rule names was more complex to test as it required a full CloG proof. Then the bullet translation test also required the addition of a CloNames relation to test with.

The automated tests of the transformation, see Section 6.4.2, are not split up. Instead, an example proof was used containing at least one application case of each of the in-directly transformed rules: clo_{x}, exp, and the closure rule. This test covers all of the important parts of the transformation and also includes many directly mapped rule cases.

The tests for LaTeX output use the same testing technique as for the syntax conversion. There are no automated tests for the GLTool module as the IDE() function has no testable output, and the other functions have very simple functionality to call the other modules that are already automatically tested.

7.2 Manual testing

As a last check of the correctness of the transformation, The complex CloG proof from Section 3.1.2 which has more than one application of the $\mathsf{clo_x}$ rule is used in a manual test. For this the proof is input using the CloG syntax, and then the transformation tool is run on it. The output of this proof is quite large and can be found in Appendix B. The resulting G proof has been checked manually for validity and the transformation was, in fact, successful.

8 Results

One of the motivations for this project was to see the way the proofs in CloG and the proofs in G relate when using the transformation. One clear observation is that the G proofs are always larger than the CloG poofs. No inference rule in a CloG proof is no longer needed in the G derivation. CloG rules are only ever mapped, moved, or expanded into more rule applications. The sequents also become larger as bullet translated labelled formulae will only ever have demonic tests inserted, rather than any parts removed. These expansions also tend to be quite significant as a demonic test of the negation of the entire list of side formulae is inserted twice for every occurrence of the fixed point formula that is expanded.

The general structure of the proofs does stay the same, apart from the split into two branches at the closure rule transformations. I found that even if the CloG proof input is the most simplified proof for a given sequent, the transformation will not necessarily give the most simplified G proof. This is partly due to the deep inference rules only being used in the case of a $\mathsf{clo_x}$ or exp rule application and are not used effectively elsewhere where they could be. This is demonstrated in the Example 8.1 where a demonic test is removed in CloG by multiple steps and therefore also in the transformed G proof, but G has the $\mathsf{mon_d^f}$ rule to perform this in one step.

Example 8.1 (Proofs for the formula $\langle q!\rangle(p \vee \neg p)$)**.**

Most simplified CloG proof:

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

G proof transformed from CloG:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p, \neg p}\text{Ax}}{p \vee \neg p}\vee}{q, p \vee \neg p}\text{weak}}{q \vee p \vee \neg p}\vee}{\langle q!\rangle(p \vee \neg p)}\,!$$

Simplified G proof:

$$\dfrac{\dfrac{\dfrac{}{p, \neg p}\ \text{Ax}}{p \vee \neg p}\ \vee}{\langle q! \rangle (p \vee \neg p)}\ \text{mon}^{\mathsf{f}}_{\mathsf{d}}$$

An observation was also made about the exp rule transformation, see Section 3.3.5. The transformation accounts for the possibility of multiple occurrences of a fixed point formula θ expanded by the bullet translation. However, in practice the occurrence of multiple of these formulas seems quite uncommon due to two reasons. One is that the clo_{x} rule, which expands a fixed point formula, is not a deep inference rule so can only be applied to a fixed point formula when it is its own element of a sequent. The other is that there are few rules in CloG that will increase the complexity of a formula going fromt he root to the leaves of a proof tree. This means that an expanded fixed point formula will not often be found often as part of a larger formula, and even less often multiple times in the same formula. While I have not yet investigated, it may be that the case of multiple θ occurences will never happen within a CloG proof, and therefore have lead to an unnecessary step in the transformation implementation.

The last observation made was from the deterministic nature of the transformation. We know that there exists an efficient proof search algorithm for CloG, and given that the transformation has a determined result, we know there must also be an efficient proof search algorithm directly for G. We can essentially use the same algorithm as for CloG using the associated G rules then in the case of the clo_{x}, exp or closure use the patterns given in the transformation. This property is something that could perhaps also be extended to an efficient proof search for Par if that transformation is also implementable.

9 Conclusion

9.1 Evaluation

In the introduction of this paper I posed the research question:

> How can we create a tool that can transform a proof formulated in CloG to a proof formulated in G?

The how this tool was implemented is discussed in Section 6, and based on the testing in Section 7 the resulting tool is successful in transforming a proof formulated in CloG to a proof formulated in G. I also set out a clear set of separate requirements that the tool would need to fulfill in order to answer the research question. I successfully implemented all of the must have requirements (R-1.1,R-1.2,R-1.3,R-1.4) and the should have requirements (R-2.1,R-2.2,R-2.3). I also successfully implemented R-3.1 and R-3.3 for the could have requirements. I however did not export the project to a portable executable for R-3.2. This was due to the fact I do not have much experience with Java compilation and execution outside of an IDE. Since I came to attempting to implement this requirement later in the time frame of my project, I could not spend the time on researching and learning without taking away from other parts of the project, so decided to not include the requirement.

I did not have many troubles when implementing the algebraic data types, syntax conversion, and LaTeX output. These were either made simple by Rascal's syntax tree tools or were some simple case by case implementations. The more difficult parts of the project came with the transformation itself. Particularly finding an efficient implementation of the transformation for a clo_{x} and exp rules added significant time to the project that caused the project to have a smaller scope. Additionally, while the algebraic type definition was straight forward, I spent more time on the concrete syntax definition. There, I had to keep the syntax unambiguous while also making the input intuitive to read and easy to write.

The tool assumes that the CloG proof given as input is valid. While the syntax is checked, there is no checking that rules are applied correctly, or in the right order. It is not checked if labels used have been defined by a clo_{x} application or if when the closure rule is used, that the context is in fact the same. If invalidates like this are included in the input then the tool will have unexpected behaviour, either throwing

an error or outputting and invalid G proof.

I implemented automated testing that covers the majority of my code. If one comes back to edit this code the automated tests will be very helpful to easily check altered functions have not stopped working correctly. However, the tests I have do not necessarily cover all edge cases and possibilities in a CloG proof. While not all cases can ever be covered, certainly more cases could be covered by larger proofs. However, currently I have needed to manually make the proofs and then manually check the validity of the returned G proof. The time this takes has meant that I have not been able to carry out the more complex tests in this project's time frame.

For any software implementation, having good usability is an important attribute. Needing to follow the Rascal instructions for installing the Eclipse IDE in order to use this tool is not ideal. However, I am happy with the way I have organised the input, output, and execution of the tool to make it easy to use for someone who does not necessarily understand Rascal or the inner workings of the tool.

9.2 Future Work

While the project has proven successful in achieving the intended scope and capability described in the requirements section, there are still plenty of improvements and additions that could be made to the tool. A static analysis tool could be added to check that the proof input is valid before attempting to carry out the transformation. Some more complex tests, automated or manual, could also be developed to ensure that the code is more robust. Compilation into a portable executable that can be used without Eclipse would also be a useful addition.

An automated theorem prover for CloG is another future project that would be useful in this area. This could be used to create test input for this tool. It could also, in combination with this tool, be used as a proof solver for G. Additionally, another transformation defined in [3] is from G to Par. In the introduction I described how Par makes for more intuitive proofs, but has a less efficient proof search strategy. If the transformation from G to Par were to also be implemented in future, these three tools in combination would then make an efficient automated theorem prover for CloG, G and Par.

An investigation into the case of the exp rule transformation discussed in Section 8, could be carried out. This may reveal that certain cases covered by the transformation will not appear in a CloG proof. Building upon the applications of the proof systems discussed in this paper, more properties like this may also be found.

Lastly, while not related to these proof systems, another future project in automation for game logic could be model check. How to find the valuation of a game logic formula is discussed in Section 2.5. This could be implemented, with the input of a game logic model, to find the set of states where a formula evaluates to true.

References

[1] A. Tubella and L. Straßburger. Introduction to deep inference. *lecture notes for ESSLLI'19. [Online]*, 2019. URL: `https://hal.inria.fr/hal-02390267/document`.

[2] M. Pauly and R. Parikh. Game logic: An overview. *Studia Logica: An International Journal for Symbolic Logic*, 75(2):165–182, 2003. URL: `http://www.jstor.org/stable/20016549`.

[3] S. Enqvist, H. H. Hansen, C. Kupke, J. Marti, and Y. Venema. Completeness for game logic. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2019. `doi:10.1109/LICS.2019.8785676`.

[4] S. Graham-Lengrand. Psyche: A proof-search engine based on sequent calculus with an lcf-style architecture. In D. Galmiche and D. Larchey-Wendling, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 149–156, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[5] M. S. Nawaz, M. Sun, and P. Fournier-Viger. Proof searching in pvs theorem prover using simulated annealing. In Y. Tan and Y. Shi, editors, *Advances in Swarm Intelligence*, pages 253–262, Cham, 2021. Springer International Publishing.

[6] A. Platzer and G. Sutcliffe. Automated deduction–cade 28: 28th international conference on automated deduction, virtual event, july 12–15, 2021, proceedings, 2021.

[7] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39(1):176–210, 1935.

[8] M. Baaz and A. Leitsch. Cut-elimination and redundancy-elimination by resolution. *Journal of Symbolic Computation*, 29(2):149–176, 2000. doi:https://doi.org/10.1006/jsco.1999.0359.

[9] S. Hetzl, A. Leitsch, D. Weller, and B. W. Paleo. Transforming and analyzing proofs in the ceres-system. In Piotr Rudnicki, Geoff Sutcliffe, Boris Konev, Renate A. Schmidt, and Stephan Schulz, editors, *Proceedings of the LPAR 2008 Workshops, Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics, Doha, Qatar, November 22, 2008*, volume 418 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008. URL: http://ceur-ws.org/Vol-418/paper6.pdf.

[10] P. Klint, T. v.d. Storm, and J. 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.

A Rascal Annotated Code

Listing 1: "GLTool.rsc"

```
module GLTool
/*
 * Main module for transformation tool
 *
 * Call CloG2G_Tool with the name of the input file to use.
 * Output .tex file can be found in the output folder.
 */

import CloGSyntax;
import GLASTs;
import CST2AST_CloG;
import LaTeXOutput;
import CloG2G;

import util::IDE;
import ParseTree;

// Call IDE() in terminal to activate parse checker and keyword hihlighting for .clog files in eclipse.
void IDE() {
        start[SCloGProof] clog(str src, loc l) {
                return parse(#start[SCloGProof], src, l);
        }



        registerLanguage("CloG", "clog", clog);
}

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

```
// Get the location of an input .clog file.
loc inputLoc(str file) {
        return (|project://Game-Logic-Proof-Transformation-Tool/input| + file)[extension=".clog"];
}

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

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

// Input abstract synta 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);
}

// Input abstract synta tree for G proof and output CloG .tex proof tree to the given output file
void G2LaTeX(GProof p, str out){
        loc l = outputLoc(out);
        LaTeXOutput(p, l);
}

// Input .clog file name and output Clooutput .tex proof tree to the given output file
void input2latex(str \in,  str out){
        CloG2LaTeX(getCloGAST(\in), out);
}

// Input .clog file name, get proof transformed to G and output G .tex proof tree to the given output file
void CloG2G_Tool(str file){
        CloGProof ast = getCloGAST(file);
        GProof gp = CloG2G(ast);
        G2LaTeX(gp, file);
}
```

Listing 2: "GLASTs.rsc"

```
module GLASTs
/*
 * Module defining all Abstract Syntax Types for the Game logic proof transformations
 */

/*
 * Abstract Syntax for a CloG Proof
 */

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);

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

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 G Proof
 */

data GProof(loc src = |tmp:///|)
        = GLeaf()
        | GUnaryInf(list[GameLog] seq, GRule rule, GProof inf)
        | GBinaryInf(list[GameLog] seq, GProof infL, GProof infR);

data GRule(loc src = |tmp:///|)
        = ax()
        | Gweak()
        | Gmodm()
        | Gor()
        | Gand()
        | Giter()
        | GdIter()
        | Gchoice()
        | GdChoice()
        | Gtest()
        | GdTest()
        | mongd()
        | monfd()
        | concatd()
        | inds();

/*
 * 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);
```

Listing 3: "CloGSyntax.rsc"

```
module CloGSyntax
/*
 * Module defines the concrete syntax for the input of a CloG proof
 *
 * Converted to abstract syntax tree by CST2AST_CloG module
 */

extend lang::std::Layout;
extend lang::std::Id;

// Proof file starts with "CloG" and wrapped with {}
start syntax SCloGProof
        = "CloG" "{" SCloGPart* ps "}";

/*
 * The CloG file consists of a list of sequents paired with either the rule they are inferred by, a discarged Clo ru
 *
 * The list of sequents are also written in the order from the bottom of a proof tree to the top.
 *
 * For the use of the conjuction rule, the proof splits into two branches that are wrapped by two sets of {}. This c
 *
 * Each sequent consists of a list of CLoG terms wrapped by [] for which the syntax is also defined.
 */
syntax SCloGPart
        = "[" SCloGTerm* seq "]" SCloGRule rule ";"
        | "[" SCloGTerm* seq "]" "and" ";" "{" SCloGPart* psL "}" "{" SCloGPart* psR "}"
        | "leaf" ";";


/*
 * 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 for list of rules defined for CloG, excluding the conjunction rule
syntax SCloGRule
        = "Ax1"
        | "modm"
        | "or"
        | "choice"
        | "dChoice"
        | "weak"
        | "exp"
        | "concat"
        | "iter"
        | "dIter"
        | "test"
        | "dTest"
        | "clo" "_" SId n
        | "closure" "^" SId n;
```

```
// 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];
```

Listing 4: "CST2AST_CloG.rsc"

```
module CST2AST_CloG
/*
 * Module containing function to transform CloG input to an abstract syntax tree
 */

import ParseTree;
import String;

import CloGSyntax;
import GLASTs;
import List;

// Main function for syntax conversion
CloGProof cst2astCloG(start[SCloGProof] sp){
	SCloGProof p = sp.top;

	return cst2astCloG([ pp | SCloGPart pp <- p.ps ]);
}

// Function for syntax conversion of a list of sequents and rules
CloGProof cst2astCloG(list[SCloGPart] ps){
	pps = headTail(ps);
	switch(pps[0]){
		case (SCloGPart)`leaf ;`:
			return CloGLeaf();
		case (SCloGPart)`[ <SCloGTerm* seq> ] closure ^ <SId n> ;`:
			return disClo([cst2astCloG(t) | SCloGTerm t <- seq ],
			                                id2name(n));
		case (SCloGPart)`[ <SCloGTerm* seq> ] <SCloGRule rule> ;`:
			return CloGUnaryInf([cst2astCloG(t) | SCloGTerm t <- seq ],
			                                       cst2astCloG(rule),
			                                       cst2astCloG(pps[1]));
		case (SCloGPart)`[ <SCloGTerm* seq> ] and ; { <SCloGPart* psL> } { <SCloGPart* psR> }`:
```

```
                        return CloGBinaryInf([cst2astCloG(t) | SCloGTerm t ← seq ],
                                                            cst2astCloG([ pp | SCloGPart pp ← psL ]),
                                                            cst2astCloG([ pp | SCloGPart pp ← psR ]));
                default: throw "Unsupported CloG Proof";
        }
}

// Function for syntax conversion of a CloG term
CloGTerm cst2astCloG(SCloGTerm t){
        return term(cst2astCloG(t.ex), [id2name(n) | SId n ← t.label]);
}

// Function for syntax converstion of a CloG rule
CloGRule cst2astCloG(SCloGRule r){
        switch(r){
                case (SCloGRule)`Ax1`: return ax1();
                case (SCloGRule)`modm`: return modm();
                case (SCloGRule)`or`: return orR();
                case (SCloGRule)`choice`: return choiceR();
                case (SCloGRule)`dChoice`: return dChoiceR();
                case (SCloGRule)`weak`: return weak();
                case (SCloGRule)`exp`: return exp();
                case (SCloGRule)`concat`: return concatR();
                case (SCloGRule)`iter`: return iterR();
                case (SCloGRule)`dIter`: return dIterR();
                case (SCloGRule)`test`: return testR();
                case (SCloGRule)`dTest`: return dTestR();
                case (SCloGRule)`clo _ <SId n>`: return clo(id2name(n));
                default: throw "Unsupported CloG Rule";
        }
}

// Function for syntax conversion of a normal form game logic formula
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
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";
        }
}

// 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";
        }
}
```

Listing 5: ”BulletTranslate.rsc”

```
module BulletTranslate
/*
 * Module to perform the bullet translation of CloG terms to game logic formulae for G proofs
 *
 * Module also used to generate the Clo rule names environment variable
 */

import Set;
import IO;
import List;
import LaTeXOutput;

import CloG2G;
import GLASTs;

// Definition of relation from a Clo rule name to the sequent information where it was applied
alias CloNames = rel[CloGName x, CloSeq seq];

// Definition of tuple containing the bullet translated side formulae, name assignemnt and associated fixed point fo
alias CloSeq = tuple[list[GameLog] side, GameLog Xx, GameLog fp];

// Function to create relation for all clo rule names in a CloG proof.
CloNames resolveCloRules(CloGProof p){
```

```
        CloNames ns = {};

        visit(p){
                case CloGUnaryInf(list[CloGTerm] seq, clo(CloGName x), CloGProof inf):{
                        ns += {<x, getCloSeq(seq, inf.seq, x, ns)>};
                }
        }
        return ns;
}

/*
 * Function to form the tuple of sequent information for a given CloG rule name
 *
 * This function also requires the use of other already resolved clo names. This is handled by
 * the order in which the previous function creates the relation, so that all necessary names will already be includ
 */
CloSeq getCloSeq(list[CloGTerm] seq, list[CloGTerm] infSeq, CloGName x, CloNames ns){
        for (/term(\mod(dIter(Game g), GameLog pr), list[CloGName] la) := seq)
                for (/term(and(pr,\mod(g,\mod(dIter(g),pr))), list[CloGName] l) := infSeq)
                        if (x in l){
                                list[CloGTerm] side = delete(seq,indexOf(seq,term(\mod(dIter(g),pr),la)));
                                return <CloG2G(side, ns), nameAssign(side, ns), \mod(dIter(g),pr)>;
                        }
        return <[], atomP(prop("")), atomP(prop(""))>;
}

// Function to negate a game logic formula and preserve its normal form.
GameLog negNF(top()) = bot();
GameLog negNF(bot()) = top();
GameLog negNF(atomP(Prop p)) = neg(atomP(p));
GameLog negNF(neg(atomP(Prop p))) = atomP(p);
GameLog negNF(or(GameLog gl, GameLog gr)) = and(negNF(gl),negNF(gr));
GameLog negNF(and(GameLog gl, GameLog gr)) = or(negNF(gl),negNF(gr));
GameLog negNF(\mod(Game ga, GameLog g)) = \mod(negNF(ga), negNF(g));

// Function to get the dual of a game formula and preserve its normal form
Game negNF(atomG(AGame g)) = dual(atomG(g));
Game negNF(dual(atomG(AGame g))) = atomG(g);
Game negNF(concat(Game gal, Game gar)) = concat(negNF(gal), negNF(gar));
Game negNF(choice(Game gal, Game gar)) = dChoice(negNF(gal), negNF(gar));
Game negNF(dChoice(Game gal, Game gar)) = choice(negNF(gal), negNF(gar));
Game negNF(iter(Game ga)) = dIter(negNF(ga));
Game negNF(dIter(Game ga)) = iter(negNF(ga));
Game negNF(\test(GameLog g)) = dTest(negNF(g));
Game negNF(dTest(GameLog g)) = \test(negNF(g));

// Function to split a concatenated game formula into multiple mod unary connectives
GameLog concatSplit(\mod(Game g, GameLog p)){
        list[Game] gcs = getConcats(g);
        return (\mod(head(gcs), p) | \mod(gc, it) | Game gc ← tail(gcs));
}

// Function the same as concatSplit but with annotation for the exp rule
GameLog concatSplitExp(\mod(Game g, GameLog p)){
        list[Game] gcs = getConcats(g);
        Game h = head(gcs);
        return ( ((dIterExp(Game ga , int n) := h) ? modExp(dIter(ga), p, n) : \mod(h, p))
                   | ((dIterExp(Game ga, int n) := gc) ? modExp(dIter(ga), p, n) : \mod(gc, it))
                   | Game gc ← tail(gcs));
}

// Function to get a list of all games that are concatenated together in a game formula
list[Game] getConcats(concat(Game gl, Game gr)) = getConcats(gr) + getConcats(gl);
list[Game] getConcats(Game g) = [g];

// Function to get the sequent information associated with a certain clo rule application
CloSeq resName(CloNames ns, CloGName x) = getOneFrom(ns[x]);
```

```
// Function to bullet translate and negate the side formulas of a clo rule application for the name assignment
GameLog nameAssign(list[CloGTerm] seq, CloNames ns)
        = negNF(
                (bullet(head(seq), ns) | or(it,bullet(t,ns)) | CloGTerm t ← tail(seq))
        ) ? top();

// Function to concatenate a list of name assigned sequents with demonic tests together in a game formula.
// Used within the bullet translation
Game seqConcat(list[CloGName] l, CloNames ns)
        = (\dTest(resName(ns, head(l)).Xx) | concat(\test(resName(ns,x).Xx),it) | CloGName x ← tail(l));

// Function to concatenate a list of name assigned sequents with demonic tests together and also with a given extra
//into another game formula. Used within the bullet translation
Game seqConcatGame(list[CloGName] l, Game g, CloNames ns)
        = (g | concat(\dTest(resName(ns,x).Xx),it) | CloGName x ← l);

// Function to get the bullet translation of a game logic formula given a list of names in a label
GameLog bullet(term(top(), list[CloGName] _), CloNames _) = top();
GameLog bullet(term(bot(), list[CloGName] _), CloNames _) = bot();
GameLog bullet(term(atomP(Prop p), list[CloGName] _), CloNames _) = atomP(p);
GameLog bullet(term(neg(atomP(Prop p)), list[CloGName] _), CloNames _) = neg(atomP(p));
GameLog bullet(term(or(GameLog gl, GameLog gr), list[CloGName] l), CloNames ns)
        = or(bullet(term(gl,l),ns),bullet(term(gr,l),ns));
GameLog bullet(term(and(GameLog gl, GameLog gr), list[CloGName] l), CloNames ns)
        = and(bullet(term(gl,l),ns),bullet(term(gr,l),ns));
GameLog bullet(term(\mod(Game ga, GameLog g), list[CloGName] l), CloNames ns)
        = concatSplit(\mod(bullet(ga,l,g,ns),bullet(term(g,l),ns)));

// Function to get the bullet translation of a game formula given a list of names in a label and a fixed point formu
Game bullet(atomG(AGame g), list[CloGName] _, GameLog _, CloNames _)
        = atomG(g);
Game bullet(dual(atomG(AGame g)), list[CloGName] _, GameLog _, CloNames _)
        = dual(atomG(g));
Game bullet(\test(GameLog ga), list[CloGName] l, GameLog g, CloNames ns)
        = \test(bullet(term(ga,l),ns));
Game bullet(\dTest(GameLog ga), list[CloGName] l, GameLog g, CloNames ns)
        = \dTest(bullet(term(ga,l),ns));
Game bullet(iter(Game ga), list[CloGName] _, GameLog _, CloNames _)
        = iter(ga);
Game bullet(concat(Game gal, Game gar), list[CloGName] l, GameLog g, CloNames ns)
        = concat(bullet(gal,l,\mod(gar,g),ns),bullet(gar,l,g,ns));
Game bullet(choice(Game gal, Game gar), list[CloGName] l, GameLog g, CloNames ns)
        = choice(bullet(gal,l,g,ns),bullet(gar,l,g,ns));
Game bullet(dChoice(Game gal, Game gar), list[CloGName] l, GameLog g, CloNames ns)
        = dChoice(bullet(gal,l,g,ns),bullet(gar,l,g,ns));

// Function to perform the bullet translation in the case of a fixed point formula
Game bullet(dIter(Game ga), list[CloGName] l, GameLog g, CloNames ns){
        list[CloGName] X = [x | CloGName x ← l, resName(ns,x).fp == \mod(dIter(ga),g)];
        return concat(dIter(seqConcatGame(X,ga,ns)),seqConcat(X,ns))
                ? dIter(ga);
}

// Functions the same as bullet but with additional annotation for the expansion rule
GameLog bulletExp(term(top(), list[CloGName] _), CloNames _, _) = top();
GameLog bulletExp(term(bot(), list[CloGName] _), CloNames _, _) = bot();
GameLog bulletExp(term(atomP(Prop p), list[CloGName] _), CloNames _, _) = atomP(p);
GameLog bulletExp(term(neg(atomP(Prop p)), list[CloGName] _), CloNames _, _) = neg(atomP(p));
GameLog bulletExp(term(or(GameLog gl, GameLog gr), list[CloGName] l), CloNames ns, CloGName x)
        = or(bulletExp(term(gl,l),ns,x),bulletExp(term(gr,l),ns,x));
GameLog bulletExp(term(and(GameLog gl, GameLog gr), list[CloGName] l), CloNames ns, CloGName x)
        = and(bulletExp(term(gl,l),ns,x),bulletExp(term(gr,l),ns,x));
GameLog bulletExp(term(\mod(Game ga, GameLog g), list[CloGName] l), CloNames ns, CloGName x)
        = concatSplitExp(\mod(bulletExp(ga,l,g,ns,x),bulletExp(term(g,l),ns,x)));
Game bulletExp(atomG(AGame g), list[CloGName] _, GameLog _, CloNames _, CloGName x)
        = atomG(g);
Game bulletExp(dual(atomG(AGame g)), list[CloGName] _, GameLog _, CloNames _, CloGName x)
        = dual(atomG(g));
```

```
Game bulletExp(\test(GameLog ga), list[CloGName] l, GameLog g, CloNames ns, CloGName x)
        = \test(bulletExp(term(ga,l),ns,x));
Game bulletExp(\dTest(GameLog ga), list[CloGName] l, GameLog g, CloNames ns, CloGName x)
        = \dTest(bulletExp(term(ga,l),ns,x));
Game bulletExp(iter(Game ga), list[CloGName] _, GameLog _, CloNames _, CloGName x)
        = iter(ga);
Game bulletExp(concat(Game gal, Game gar), list[CloGName] l, GameLog g, CloNames ns, CloGName x)
        = concat(bulletExp(gal,l,\mod(gar,g),ns,x),bulletExp(gar,l,g,ns,x));
Game bulletExp(choice(Game gal, Game gar), list[CloGName] l, GameLog g, CloNames ns, CloGName x)
        = choice(bulletExp(gal,l,g,ns,x),bulletExp(gar,l,g,ns,x));
Game bulletExp(dChoice(Game gal, Game gar), list[CloGName] l, GameLog g, CloNames ns, CloGName x)
        = dChoice(bulletExp(gal,l,g,ns,x),bulletExp(gar,l,g,ns,x));
Game bulletExp(dIter(Game ga), list[CloGName] l, GameLog g, CloNames ns, CloGName x){
        list[CloGName] X = [x | CloGName x ← l, resName(ns,x).fp == \mod(dIter(ga),g)];
        return concat(x in X ? dIterExp(seqConcatGame(X,ga,ns),0) : dIter(seqConcatGame(X,ga,ns)),seqConcat(X,ns))
                ? dIter(ga);
}
```

Listing 6: "CloG2G.rsc"

```
module CloG2G
/*
 * Module containing the functions that perform the transformation of a CloG proof tree to a G proof tree
 */

import IO;
import List;

import LaTeXOutput;
import GLASTs;
import BulletTranslate;

// Function that first gets the clo rule names environment variable then calls the rest of the translation given the
GProof CloG2G(CloGProof p){
        CloNames ns = resolveCloRules(p);
        return CloG2G(p, ns);
}

// Case of a proof brach leaf
GProof CloG2G(CloGLeaf(), CloNames _) = GLeaf();

/*
 * Case of a discharched application of the clo rule. Identifies the active formula and then
 * builds the end of the branch given the pattern described by the transformation
 */
GProof CloG2G(disClo(list[CloGTerm] seq, CloGName x), CloNames ns) {
        CloSeq s = resName(ns, x);
        for (CloGTerm t ← seq)
                if      (!isEmpty(t.label) && last(t.label) == x && \mod(dIter(concat(Game _,Game X)),\mod(Game _,Ga
                        return GUnaryInf(s.side + \mod(dIter(concat(dTest(s.Xx),X)), \mod(dTest(s.Xx),p)), GdIter(),
                                GBinaryInf(s.side + and(\mod(dTest(s.Xx),p),\mod(concat(dTest(s.Xx),X),\mod(dIter
                                GUnaryInf(s.side + \mod(dTest(s.Xx),p), GdTest(),
                                GUnaryInf(s.side + or(s.Xx,p), Gor(),
                                GUnaryInf(s.side + s.Xx + p, Gweak(),
                                toTopAx(s.side, s.Xx)))),
                                GUnaryInf(s.side + \mod(concat(dTest(s.Xx),X),\mod(dIter(concat(dTest(s.Xx),X)),
                                GUnaryInf(s.side + \mod(dTest(s.Xx), \mod(X,\mod(dIter(concat(dTest(s.Xx),X)), \m
                                GUnaryInf(s.side + or(s.Xx, \mod(X,\mod(dIter(concat(dTest(s.Xx),X)), \mod(dTest(
                                GUnaryInf(s.side + s.Xx + \mod(X,\mod(dIter(concat(dTest(s.Xx),X)), \mod(dTest(s.
                                toTopAx(s.side, s.Xx)))))));
        return GLeaf();
}

// Function to add an extra application of the disjunction rule in the case that the expanded clo rule had no side f
GProof toTopAx([], top())
        = GUnaryInf([top()], Gor(), GUnaryInf([atomP(prop("p")), neg(atomP(prop("p")))], ax(), GLeaf()));
GProof toTopAx(list[GameLog] side, GameLog Xx)
        = GUnaryInf(side + Xx, ax(), GLeaf());
```

```
/*
 * Case of an application of the Clo rule. Identifies the active formula then calls a recursive helper formula
 * to form the given pattern described by the transformation
 */
GProof CloG2G(CloGUnaryInf(list[CloGTerm] seq, clo(CloGName x), CloGProof inf), CloNames ns){
        CloSeq s = resName(ns, x);
        CloGTerm t = getOneFrom(seq - inf.seq);
        CloGTerm ti = getOneFrom(inf.seq - seq);
        GameLog p = bullet(term(s.fp.pr, t.label), ns);
        return CloGclo2G(bullet(t,ns),t.label,bullet(ti,ns),inds(),inf,s,p,ns);
}

/*
 * Set of recursive functions that each apply one rule from the Clo rule transformation pattern and then call the ne
 */

// Function to give the induction rule application part of the Clo transformation
GProof CloGclo2G(\mod(dIter(Game XY), GameLog Xp), list[CloGName] l, GameLog ti, inds(), CloGProof inf, CloSeq s, Ga
        = GUnaryInf(s.side + \mod(dIter(XY), Xp), inds(),
          CloGclo2G(and(indLGame(p,l,ns),\mod(XY,\mod(dIter(concat(dTest(s.Xx),XY)),\mod(dTest(s.Xx),Xp)))),ti,monfd

// Function to get the formula pattern when the G induction rule is applied
GameLog indLGame(GameLog p, list[CloGName] l, CloNames ns)
        = \mod(dTest(resName(ns, head(l)).Xx), indLGame(p,tail(l),ns)) ? p;

// Function to apply the necessary number of monfd rules for the Clo transformation
GProof CloGclo2G(and(GameLog L, GameLog R), GameLog ti, monfd(), CloGProof inf, CloSeq s, GameLog p, CloNames ns){
        if (and(L,R) == ti) return CloG2G(inf, ns);
        if (L == p) return CloGclo2G(and(L,R),ti,mongd(),inf,s,ns);
        if (\mod(Game _,GameLog Xp) := L) return GUnaryInf(s.side + and(L,R), monfd(),
                                                                                                              CloG
        return CloG2G(inf, ns);
}

// Function to apply the necessary number of mongd rules for the Clo transformation
GProof CloGclo2G(and(GameLog L, GameLog R), GameLog ti, mongd(), CloGProof inf, CloSeq s, CloNames ns){
        if (and(L,R) == ti) return CloG2G(inf, ns);
        if (R.g == s.fp.g.ga) return CloGclo2G(and(L,R),ti,concatd(),inf,s,ns);
        if (\mod(concat(Game _, Game XY),GameLog g) := R) return GUnaryInf(s.side + and(L,R),monfd(),

        return CloG2G(inf, ns);
}

// Function to apply the necessary number of concatenation rules for the Clo transformation
GProof CloGclo2G(and(GameLog L, GameLog R), GameLog ti, concatd(), CloGProof inf, CloSeq s, CloNames ns){
        if (and(L,R) == ti) return CloG2G(inf, ns);
        list[Game] gcs = getConcats(R.g);
        return (CloG2G(inf, ns) | GUnaryInf(s.side + and(L,concatForm(R.pr, gcs, i)), concatd(), it) | int i ←
[0..size(tail(gcs))]);
}

// Function to form the active formula after a given number of concatenation rule appliactions in the clo transforma
GameLog concatForm(GameLog p, list[Game] gcs, int i){
        list[Game] ms = take(i, gcs);
        list[Game] cs = gcs - ms;
        return \mod((head(cs) | concat(c,it) | Game c ← tail(cs)), (p | \mod(m,it) | Game m ← ms));
}

/*
 * Case of an application of the exp rule. Performs a bullet translation with annotations to track where the formula
 */
GProof CloG2G(CloGUnaryInf(list[CloGTerm] seq, exp(), CloGProof inf), CloNames ns){
        CloGTerm t = getOneFrom(seq - inf.seq);
        CloGTerm ti = getOneFrom(inf.seq - seq);
        list[GameLog] side = CloG2G(seq - t, ns);
        CloGName x = getOneFrom(t.label - ti.label);
```

```
        GameLog p = bulletExp(t, ns, x);
        int count = 0;
        p = visit(p){
                case modExp(Game ga, GameLog g, 0):{
                        count = count + 1;
                        insert modExp(ga, g, count);
                }
        }

        // for each expansion we apply both mongd and monfd, Removing all annotations in the output at each step.
        GProof exp = GLeaf();
        Game \test = dTest(resName(ns, x).Xx);
        for (step ← [1..count+1]){
                exp = appendBranch(exp, GUnaryInf(side + removeAnno(p), mongd(), GLeaf()));
                p = visit(p){
                        case modExp(dIter(Game Xy), GameLog Xp, step) ⇒ modExp(dIter(mongdExp(Xy, \test)), Xp, step)
                }
                exp = appendBranch(exp, GUnaryInf(side + removeAnno(p), monfd(), GLeaf()));
                p = visit(p){
                        case modExp(dIter(Game Xy), GameLog Xp, step) ⇒ modExp(dIter(mongdExp(Xy, \test)), Xp, step)
                }
        }

        exp = appendBranch(exp, CloG2G(inf, ns));
        return exp;
}

// function to change a formula based on the mongd rule application
Game mongdExp(concat(Game X, Game Xy), Game \test) = X == \test ? Xy : concat(X, mongdExp(Xy, \test));

// function to change a formula based on the monfd rule application
GameLog monfdExp(\mod(Game X,GameLog Xp), Game \test) = X == \test ? Xp : \mod(X, monfdExp(Xp, \test));

// function to remove all bullet translation annotations from a formula
GameLog removeAnno(GameLog p) = visit(p){ case modExp(ga, GameLog g, _) ⇒ \mod(ga,g) };

// function adds a proof tree onto the leaves of a proof tree
GProof appendBranch(GProof p, GProof b) = visit(p){ case GLeaf() ⇒ b };

// Case of an application of the axiom rule, doesn't need to recursively call the transformation function again
GProof CloG2G(CloGUnaryInf(list[CloGTerm] seq, ax1(), CloGProof inf), CloNames ns)
        = GUnaryInf(CloG2G(seq, ns), ax(), GLeaf());

// Case of an application of the conjunction rule splits and recursively calls the transformation on the two branche
GProof CloG2G(CloGBinaryInf(list[CloGTerm] seq, CloGProof infL, CloGProof infR), CloNames ns)
        = GBinaryInf(CloG2G(seq, ns), CloG2G(infL, ns), CloG2G(infR, ns));

// Default case for a trivial rule application that maps directly to another rule in G
GProof CloG2G(CloGUnaryInf(list[CloGTerm] seq, CloGRule rule, CloGProof inf), CloNames ns){
        list[GameLog] thisSeq = CloG2G(seq, ns);
        list[GameLog] nextSeq = CloG2G(inf.seq, ns);
        if (thisSeq == nextSeq) return CloG2G(inf, ns);
        else return GUnaryInf(CloG2G(seq, ns), CloG2G(rule), CloG2G(inf, ns));
}

// Bullet translation of all formulas in a given sequent
list[GameLog] CloG2G(list[CloGTerm] seq, CloNames ns)
        = [bullet(t, ns) | CloGTerm t ← seq];

// Function to get the G rule from a CloG rule when there is a trivial direct map
GRule CloG2G(modm()) = Gmodm();
GRule CloG2G(andR()) = Gand();
GRule CloG2G(orR()) = Gor();
GRule CloG2G(choiceR()) = Gchoice();
GRule CloG2G(dChoiceR()) = GdChoice();
GRule CloG2G(weak()) = Gweak();
GRule CloG2G(concatR()) = concatd();
GRule CloG2G(iterR()) = Giter();
```

```
GRule CloG2G(testR()) = Gtest();
GRule CloG2G(dIterR()) = GdIter();
GRule CloG2G(dTestR()) = GdTest();
```

Listing 7: "LaTeXOutput.rsc"

```
module LaTeXOutput
/*
 * Module defining the transformation from abstract proof trees to LaTeX proof trees
 */

import 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));
}
void LaTeXOutput(GProof 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}";
str LaTeXOutput(GProof p) =
	"\\documentclass{article}
	'\\usepackage[utf8]{inputenc}
	'\\usepackage{prooftrees}
	'
	'\\begin{document}
	'
	'\\begin{prooftree}
	'<LaTeXGTree(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{$[<(LaTeXCloGTree(head(seq)) | it + ", " + LaTeXCloGTree(t) | CloGTerm t ← tail(seq))>]^{<LaTeXClo
str LaTeXCloGTree(CloGUnaryInf(list[CloGTerm] seq, CloGRule rule, CloGProof inf)) =
	"<LaTeXCloGTree(inf)>
	'\\RightLabel{<LaTeXCloGTree(rule)>}
	'\\UnaryInfC{$<(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
str LaTeXCloGTree(term(GameLog s, []))
	= "(<LaTeXGameLog(s)>)^{\\varepsilon}";
```

```
str LaTeXCloGTree(term(GameLog s, list[CloGName] label))
        = "(<LaTeXGameLog(s)>)^{<(LaTeXCloGTree(head(label)) | it + ", " + LaTeXCloGTree(n) | CloGName n ←
tail(label))>}";

// 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 LaTeX proof tree part for each G sequent and assocated rule label
str LaTeXGTree(GLeaf())
        = "\\AxiomC{}";
str LaTeXGTree(GUnaryInf(list[GameLog] seq, GRule rule, GProof inf)) =
        "<LaTeXGTree(inf)>
        '\\RightLabel{<LaTeXGTree(rule)>}
        '\\UnaryInfC{$<(LaTeXGameLog(head(seq)) | it + ", " + LaTeXGameLog(g) | GameLog g ← tail(seq))>$}";
str LaTeXGTree(GBinaryInf(list[GameLog] seq, GProof infL, GProof infR)) =
        "<LaTeXGTree(infL)>
        '<LaTeXGTree(infR)>
        '\\RightLabel{$\\wedge$}
        '\\BinaryInfC{$<(LaTeXGameLog(head(seq)) | it + ", " + LaTeXGameLog(g) | GameLog g ← tail(seq))>$}";

// Function to output the G rule labels
str LaTeXGTree(ax()) = "Ax";
str LaTeXGTree(Gmodm()) = "mod$_{m}$";
str LaTeXGTree(Gand()) = "$\\wedge$";
str LaTeXGTree(Gor()) =  "$\\vee$";
str LaTeXGTree(Gchoice()) =  "$\\sqcup$";
str LaTeXGTree(GdChoice()) =  "$\\sqcap$";
str LaTeXGTree(Gweak()) =  "weak";
str LaTeXGTree(concatd()) =  "$;_d$";
str LaTeXGTree(Giter()) =  "$*$";
str LaTeXGTree(Gtest()) =  "$?$";
str LaTeXGTree(GdIter()) =  "$\\times$";
str LaTeXGTree(GdTest()) =  "$!$";
str LaTeXGTree(mongd()) =  "mon$^g_d$";
str LaTeXGTree(monfd()) =  "mon$^f_d$";
str LaTeXGTree(inds()) =  "ind$_s$";

// 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>}";
```

B Manual Test Output G Proof

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q}\ \text{Ax}$$

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q,\ \langle a\rangle\langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)^\times\rangle\langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \text{weak}$$

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q\vee\langle a\rangle\langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)^\times\rangle\langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \vee$$

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle a\rangle\langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)^\times\rangle\langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ !$$

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)\rangle\langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)^\times\rangle\langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ ;_d$$

$$*_7$$

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q}\ \text{Ax}$$

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q,\ \langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \text{weak}$$

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q\vee\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \vee$$

$$\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q\ \ !\qquad *_7$$

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q\wedge}\ \wedge$$

$$\overline{\langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)\rangle\langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)^\times\rangle\langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \times$$

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)^\times\rangle\langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \text{mod}_m$$

$$\overline{\langle a^d\rangle\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a\rangle\langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)^\times\rangle\langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \text{weak}$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a^d\rangle\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a\rangle\langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)^\times\rangle\langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \vee$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q\vee\langle a^d\rangle\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a\rangle\langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)^\times\rangle\langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ *$$

$$\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a\rangle\langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)^\times\rangle\langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q$$

$$*_6$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(a^\times\sqcap b^\times)^\times\rangle q}\ \text{Ax}$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(a^\times\sqcap b^\times)^\times\rangle q,\ q}\ \text{weak}$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(a^\times\sqcap b^\times)^\times\rangle q\vee q}\ \vee$$

$$\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q\ \ !$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(a^\times\sqcap b^\times)^\times\rangle q}\ \text{Ax}$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(a^\times\sqcap b^\times)^\times\rangle q,\ \langle((p?;a^\times)\sqcup(\neg p?;b^\times))\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \text{weak}$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(a^\times\sqcap b^\times)^\times\rangle q\vee\langle((p?;a^\times)\sqcup(\neg p?;b^\times))\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \vee$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle((p?;a^\times)\sqcup(\neg p?;b^\times))\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ !$$

$$\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q\ \ ;_d$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q\wedge\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \wedge$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \times$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a^d\rangle\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \text{weak}$$

$$\overline{\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q\vee\langle a^d\rangle\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \vee$$

$$\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q\ \ *\qquad *_6$$

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q\wedge}\ \wedge$$

$$\overline{\langle a\rangle\langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)^\times\rangle\langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \text{mon}_d^f$$

$$\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q\wedge$$

$$\overline{\langle a\rangle\langle(\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;a)^\times\rangle\langle\langle a^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \text{ind}_s$$

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a^\times\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \text{weak}$$

$$\overline{\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a^\times\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \text{weak}$$

$$\langle a^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle a^\times\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q,$$
$$\neg p\wedge\langle b^\times\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q$$

$$*_2$$

$$\overline{\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q}\ \text{Ax}$$

$$\overline{\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q,\ \langle b\rangle\langle(\langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;b)^\times\rangle\langle\langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \text{weak}$$

$$\overline{\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q\vee\langle b\rangle\langle(\langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;b)^\times\rangle\langle\langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ \vee$$

$$\overline{\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle\langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle b\rangle\langle(\langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;b)^\times\rangle\langle\langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ !$$

$$\overline{\langle b^{d^*}\rangle\langle(a^{d^*}\sqcup b^{d^*})^*\rangle\neg q,\ \langle(\langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;b)\rangle\langle(\langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!;b)^\times\rangle\langle\langle b^\times\rangle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle\langle(\langle(a^\times\sqcap b^\times)^\times\rangle q!;((p?;a^\times)\sqcup(\neg p?;b^\times)))^\times\rangle\langle\langle(a^\times\sqcap b^\times)^\times\rangle q!\rangle q}\ ;_d$$

$$*_5$$

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q$$

weak

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q,\ \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

$\vee$

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q \vee \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

!

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle \langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q \qquad *_5$$

$\wedge$

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle \langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q \wedge$$
$$\langle (\langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; b) \rangle \langle (\langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; b)^{\times} \rangle \langle \langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

$\times$

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (\langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; b)^{\times} \rangle \langle \langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

mod_m

$$\langle b^{d} \rangle \langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b \rangle \langle (\langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; b)^{\times} \rangle \langle \langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

weak

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b^{d} \rangle \langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b \rangle \langle (\langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; b)^{\times} \rangle \langle \langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

$\vee$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q \vee \langle b^{d} \rangle \langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg p,\ \langle b \rangle \langle (\langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; b)^{\times} \rangle \langle \langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

$*$

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b \rangle \langle (\langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; b)^{\times} \rangle \langle \langle b^{\times} \rangle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

$*_4$

Left branch:

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q$$

weak

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q, q$$

$\vee$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q \vee q$$

!

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

Right branch:

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q$$

weak

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q,\ \langle ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})) \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

$\vee$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q \vee \langle ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})) \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

!

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle \langle ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})) \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

$;_d$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times}))) \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

Conclusion:

$\wedge$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q \wedge \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times}))) \rangle \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

$\times$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

weak

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b \rangle \langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

$\vee$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q \vee \langle b \rangle \langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

$*$

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (\langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q!; ((p?; a^{\times}) \sqcup (\neg p?; b^{\times})))^{\times} \rangle \langle \langle (a^{\times} \sqcap b^{\times})^{\times} \rangle q! \rangle q$$

$*_3$

$$*_3 \qquad *_4$$

$\wedge$

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q \wedge$$
$$\langle b \rangle \langle (\langle b^\times \rangle \langle (a^\times \sqcap b^\times)^\times \rangle q!; b)^\times \rangle \langle \langle b^\times \rangle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

mon_d^f

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q \wedge$$
$$\langle b \rangle \langle (\langle b^\times \rangle \langle (a^\times \sqcap b^\times)^\times \rangle q!; b)^\times \rangle \langle \langle b^\times \rangle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

ind_s

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

weak

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ p,\ \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

$$\overline{p, \neg p}\ \text{Ax}$$

weak

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ p,\ \neg p$$

$\wedge$

$$\langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ p,\ \neg p \wedge \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

weak

$$\langle a^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ p,\ \neg p \wedge \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

$$*_1$$

$$*_1 \qquad *_2$$

$\wedge$

$$\langle a^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ p \wedge \langle a^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q,$$
$$\neg p \wedge \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

?

$$\langle a^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ p \wedge \langle a^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q,$$
$$\langle \neg p? \rangle \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

?

$$\langle a^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle p? \rangle \langle a^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q,$$
$$\langle \neg p? \rangle \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

$\vee$

$$\langle a^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q \vee \langle b^{d^*} \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle p? \rangle \langle a^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q,$$
$$\langle \neg p? \rangle \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

$\sqcup$

$$\langle (a^{d^*} \sqcup b^{d^*}) \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle p? \rangle \langle a^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q,\ \langle \neg p? \rangle \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

weak

$$\neg q,\ \langle (a^{d^*} \sqcup b^{d^*}) \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle p? \rangle \langle a^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q,\ \langle \neg p? \rangle \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

$\vee$

$$\neg q \vee \langle (a^{d^*} \sqcup b^{d^*}) \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle p? \rangle \langle a^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q,\ \langle \neg p? \rangle \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

$*$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle p? \rangle \langle a^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q,\ \langle \neg p? \rangle \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

$\vee$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle p? \rangle \langle a^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q \vee \langle \neg p? \rangle \langle b^\times \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

$\sqcup$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle ((p?; a^\times) \sqcup (\neg p?; b^\times)) \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

$$*_0$$

$$\overline{\neg q, q}\ \text{Ax}$$

weak

$$\neg q,\ \langle (a^{d^*} \sqcup b^{d^*}) \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ q$$

$\vee$

$$\neg q \vee \langle (a^{d^*} \sqcup b^{d^*}) \rangle \langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ q$$

$*$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ q \qquad *_0$$

$\wedge$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ q \wedge \langle ((p?; a^\times) \sqcup (\neg p?; b^\times)) \rangle \langle (\langle (a^\times \sqcap b^\times)^\times \rangle q!; ((p?; a^\times) \sqcup (\neg p?; b^\times)))^\times \rangle \langle \langle (a^\times \sqcap b^\times)^\times \rangle q! \rangle q$$

ind_s

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q,\ \langle ((p?; a^\times) \sqcup (\neg p?; b^\times))^\times \rangle q$$

$\vee$

$$\langle (a^{d^*} \sqcup b^{d^*})^* \rangle \neg q \vee \langle ((p?; a^\times) \sqcup (\neg p?; b^\times))^\times \rangle q$$