Master Thesis

Algorithms for Context-free Games: A Comparison of Saturation, Guess & Check, and Summarization

Elisabeth Neumann

September 27, 2017

Department of Computer Science
University of Kaiserslautern

First reviewer:	Prof. Dr. Klaus Schneider
Second reviewer:	Prof. Dr. Roland Meyer
Supervisor:	M. Sc. Sebastian Muskalla

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich durch die Angabe der Quelle, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht.

Kaiserslautern, den 27. September 2017

Elisabeth Neumann

Abstract

Program synthesis is a natural extension of program verification. Instead of verifying an already written program, the synthesis algorithm automatically generates the program from a program template and a specification. For recursive programs, we can model program synthesis by a two-player context-free grammar game. The program template is represented by a context-free grammar with ownership partition of the non-terminals. The non-terminals represent program locations, if the synthesis is in control, the corresponding non-terminal belongs to player prover, otherwise it belongs to the adversary player refuter. The specification is given by the language of a finite automaton. Player prover aims to derive a terminal word contained in the language of the automaton, refuter tries to prevent inclusion. The goal is to determine whether prover has a winning strategy.

We present three methods to solve context-free games and prove their correctness. In the summary approach [5], the plays of the context-free game are finitely represented by formula summaries, computed by a Kleene iteration over a system of equations derived from the grammar rules. In the saturation approach [1], we compute the pre*-image of the terminal accepted by the automaton by saturating an alternating automaton. In the Guess & Check approach [10], we reduce the context-free game to a reachability game on a finite graph by replacing the derivations by a series of guess and checks. The winner is determined by a attractor construction.

Finally, we present a detailed comparison of the three methods. We are mainly interested in the intermediary solutions that the three methods compute during the fixed-point iterations in their algorithms (Kleene iteration, saturation and attractor). We show that there is a strong relation between the formula summaries, the transitions of the saturated automaton and the nodes from the finite game that are contained in the attractor.

Contents

1. Introduction

Due to the error-prone nature of programming, programmers make extensive use of program verification to find errors in the code. However producing correct program code amounts to iterating the steps of verifying the program and fixing the bugs that were found by the verification. The idea of program synthesis is to avoid this iteration altogether by directly producing correct program code. A synthesis algorithm takes as input a program template and a specification. A program template can be understood as program code with missing expressions. The synthesis algorithm tries to fill the gaps with program code s.t. the resulting program satisfies the specification. If no such completed program exists, the synthesis algorithm should return false.

Formally, we state the **program synthesis problem** as follows. Given a program template T and a specification ϕ, does there exist an instantiation $T@i$ of T satisfying the specification ϕ.

In this thesis, we consider recursive programs with read expressions. This means the program template may read a value from an external source, for example the memory, and make decisions based on this value. In Figure 1 in the left program, we are in such a situation. The program reads a value and stores it in variable. Depending on the value of the variable, either function $H_1()$ or $H_2()$ is called. The program synthesis has to react to both possibilities and ensure that in either case the specification is satisfied. The program on the right-hand side of Figure 1 depicts a situation where the synthesis is allowed to fill in code and thereby decide which function is called.

Therefore the program template contains two types of non-determinism. On one hand, we have the angelic non-determinism, which the synthesis can control (right program). On the other hand there is the demonic non-determinism, which the synthesis can not control and has to react to (left program).

We model the program synthesis as a two-player grammar game, called context-free game. The first player is called refuter $\bigcirc$ and she is allowed to resolve the demonic non-determinism in the program. The second player is called prover ($\square$). She is allowed to control the angelic non-determinism. Prover wins a play in the game if the program from the choices of the players satisfies the specification. Prover wins the whole game if she has a strategy s.t. all programs resulting from the synthesis satisfy ϕ, no matter what refuter does. This strategy represents the instantiation $T@i$ we try to determine in the synthesis problem.

The program template is represented by a context-free grammar G with ownership partition of the non-terminals. The non-terminals H_1, H_2 represent calls of functions and the terminal words w derivable from a non-terminal represent the possible traces of the function. The right-hand side of a grammar rule describes the body of the function corresponding to the left-hand side of the rule. If there are several rules with the same left-hand side F, the function contains some kind of non-determinism. If F belongs to prover, there is an angelic non-deterministic choice in the corresponding function. Therefore, in the example program on the right of Figure 1, the non-terminal F would belong to prover. For the left program, the non-terminal F would belong to refuter.

A play of the context-free game corresponds to a derivation process in the context-free grammar G. The player owning the non-terminal is allowed to choose the grammar rule that is applied in the derivation process. We only consider left-most derivation, which corresponds to a sequential execution of the program. We can consider the context-free game as a game on an infinite graph, where the positions are given by the sentential forms and the moves by the left-derivation relation.

We represent the specification by the language $\mathcal{L}(A)$ of a finite automaton A. Prover wins the context-free game $\mathcal{G}$ given by the grammar G and the automaton A if she either has a strategy to derive a terminal word $w \in \mathcal{L}(A)$ or to enforce an infinite play. Otherwise, refuter wins. As refuter only wins finite plays ending at words $w \notin \mathcal{L}(A)$, she plays a reachability game.

```
proc F()
    x=read(source)
    if (x==0)
        $H_1$()
    else
        $H_2$()
```

```
proc F()
    if (???)
        $H_1$()
    else
        $H_2$()
```

$F_{\bigcirc} \to_G H_1 \mid H_2$ $F_{\Box} \to_G H_1 \mid H_2$

Figure 1.: Demonic (left) and angelic (right) non-determinism in a program.

In this thesis, we first present three methods to solve the synthesis problem and then conduct a detailed comparison of them. The three methods that we consider are the summary, saturation and Guess & Check approach.

In the summary approach, we compute the effect of the functions in terms of their input-output relation. In terms of our model, we compute for each non-terminal a finite representation of the derivable terminal words while taking the alternating structure of the game, i.e. the choices of the players, into account. As there may be an infinite number of words derivable from a non-terminal, we represent the effects of these words on the automaton A instead. By effects we mean state changes induced by the word, thus there can be only finitely many. To compute the summaries, we set up a system of equations closely reflecting the grammar rules and compute its least solution by Kleene iteration. To determine the winner of the game, we analyze the finite representation of the words derivable from the starting symbol S of the grammar.

In the saturation approach, we compute the pre*-image of the sentential forms that appear in derivation processes/plays of G. For the terminal words accepted by A, we compute a finite representation of the sentential forms preceding them in plays under consideration of the alternating structure. At the end, we verify whether the initial symbol S is among them. We start with an automaton recognizing the terminal words accepted by $\mathcal{L}(A)$. Then, we add further transitions to the automaton, i.e. we saturate the automaton, s.t. also the sentential forms preceding the terminal words in the plays are accepted by the automaton. As an automaton can only have a finite amount of transitions, the saturation of the automaton terminates. If the starting symbol S is accepted by the saturated automaton, prover wins, otherwise refuter wins.

In the Guess & Check approach, we reduce the context-free game to a game on a finite graph. The positions are the sentential forms of the grammar, a position belongs to the player owning the left-most non-terminal in the sentential form and moves correspond to application of rules. As the positions of the game arena corresponds to the sentential forms, it is usually infinite. In the Guess & Check approach, we change the course of the game to be able to store only parts of the sentential forms in the positions of the game graph. At a sentential form $wX\alpha$ in a play, refuter makes a prediction about the outcome of the play from X. Prover has two possibilities at this point. Either, she does not trust the prediction and asks to verify it. In this case, the derivation process from X is played out. If it ends according to the prediction, refuter wins the whole game, otherwise prover wins. For this part, we do not need w or α and can discard it. If prover chooses to trust the prediction, the derivation process from X is skipped completely and the play continues from a sentential form $w'\alpha$ picked by prover. Instead of the terminal words w and w', we simply track the induced state changes in A. This Guess & Check mechanic allows us to bound the size of the sentential forms stored in each node. The result is a game on a finite graph, where prover wins the original game if and only if she wins in the game on the finite graph. We can determine the winner of the finite game using the well-known attractor construction.

In the second part of the thesis, we compare the three methods. It is an interesting question whether these three approaches, while solving the same problem, also have common design principles or whether and how they differ. Furthermore, this allows us to gain insights into the mechanics of the approaches. As all three approaches include some form of fixed-point iteration (Kleene iteration for the summary approach, saturation of the automaton and attractor construction for the Guess & Check approach), we are mostly interested in whether the intermediary information computed in the iterations coincides in some way.

We show that there is indeed a strong relation between the representation of the derivable words (summary method), the transitions of the saturated automaton (saturation method) and the nodes from the finite game that are contained in the attractor (Guess & Check approach).

Related work The summary approach has been presented by Holík, Meyer and Muskalla in [5]. The saturation and the Guess & Check approach are adapted from the techniques of Cachat [1] resp. Walukiewicz [10] that solve pushdown games.

A further result from [10] allows us to translate a context-free game to a model-checking problem of a pushdown system against a μ-calculus formula. This reduction allows us to use algorithms for model-checking to solve context-free games. In [7] for example, an automata-theoretic approach to this model-checking problem is presented. In [6], Ong and Kobayashi solve the problem whether a higher order recursion schemes, which are a generalization of context-free grammars, is accepted by an alternating parity tree automaton. The alternation in the automaton can be understood as a two-player game.

Finally, Muscholl et al [9] consider a type of grammar game where one player is allowed to choose a non-terminal from the sentential forms in the derivation process and the other player is allowed to pick the grammar rule. The authors focus on the complexity and decidability of these games. Our context-free games are a special case of the games considered in [9].

2. Context-free games

In this section, we formally define the context-free games modeling the synthesis problem. A context-free game $\mathscr{G} = (G, A)$ is given by a context-free grammar G with ownership partition of the non-terminals and a finite automaton A.

A **context-free grammar** is a tuple $G = (N, T, S, \rightarrow_G)$, where N is the finite set of non-terminals, T is the finite set of terminals, $S \in N$ is the starting symbol and $\rightarrow_G \subseteq N \times (T \cup N)^*$ is the finite set of rules. We call the set of words $\mathscr{S} = (T \cup N)^*$ the sentential forms of the grammar.

In some sections, we need to address different parts of a sentential form $vX\alpha$. Therefore, we introduce the following notation. Let $vX\alpha$ be a sentential form with $v \in T^*$, $X \in N$ and $\alpha \in \mathscr{S}$. We call v the **terminal prefix**, X the **leftmost non-terminal** and α the **suffix** of the sentential form.

By $\Rightarrow\ \subseteq \mathscr{S}^* \times \mathscr{S}^*$ we denote the **left derivation relation** based on $\rightarrow_G$. It is defined by $wX\alpha \Rightarrow w\beta\alpha$ if $X \rightarrow_G \beta$ for some terminal word $w \in T^*$, non-terminal X and sentential forms $\alpha, \beta \in \mathscr{S}$. By $\Rightarrow^*$, we denote the reflexive and transitive closure of $\Rightarrow$, which is defined by $\alpha \Rightarrow^* \alpha$ and $\alpha \Rightarrow^* \beta$ if there exists an α' s.t. $\alpha \Rightarrow \alpha' \Rightarrow^* \beta$.

We call $\alpha \Rightarrow^* \beta$ a **derivation process** from α to β. We identify a derivation process π by its intermediary sentential forms, i.e. $\pi = \alpha, \alpha_1, \alpha_2, \ldots, \alpha_n, \beta$. Note that there may be several different derivation processes from α to β, as we do not require the grammar to be unambiguous.

The **language of a grammar** G is given by $\mathcal{L}(G) = \{w \in T^* \mid S \Rightarrow^* w\}$.

Example. Consider for example the following context-free grammar $\mathcal{G}_{ex}$.

$$\begin{aligned} S &\rightarrow_G c \mid XY, \\ X &\rightarrow_G a \mid aX, \\ Y &\rightarrow_G b \mid c, \end{aligned}$$

The language of $\mathcal{G}_{ex}$ is $\mathcal{L}(\mathcal{G}_{ex}) = \big(c \mid (a^*(b \mid c))\big)$.

A **finite automaton** is a tuple $A = (T, Q, q_0, Q_F, \rightarrow)$, where T is the finite input alphabet, Q is the finite set of states, $q_0 \in Q$ is the starting symbol, $Q_F \subseteq Q$ is the set of final states and $\rightarrow \subseteq Q \times T \times Q$ is the finite set of transitions or edges. Note that the input alphabet coincides with the terminals from the grammar.

We lift the definition of $\rightarrow$ from terminals to words by $q \xrightarrow{\varepsilon} q$ and $q \xrightarrow{aw} p$ if there exist a state q' s.t. $q \xrightarrow{a} q' \xrightarrow{w} q$.

We call $q \xrightarrow{w} p$ a run of w from q to p. If $q = q_0$ and $p \in Q_F$, we call the run accepting. We do not require the automaton A to be deterministic. Thus, there may be two (or more) states $p, p' \in Q$ s.t. $q \xrightarrow{w} p$ and $q \xrightarrow{w} p'$.

The **language of a finite automaton** A is given by $\mathcal{L}(A) = \{w \in T^* \mid \exists\ q_0 \xrightarrow{w} q_f \in Q_F\}$

$\mathcal{A}_{ex}$ is an example of a finite automaton. Its language is given by $\mathcal{L}(\mathcal{A}_{ex}) = \big((a^{2n}b) \mid (a^{2n+1}c)\big)$ for $n \geq 0$.

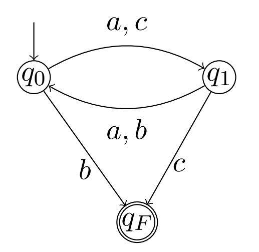


Figure 2.: Finite automaton $\mathcal{A}_{ex}$.

As already mentioned in the introduction, we model the synthesis as a two-player inclusion game. The grammar represents the program template. Therefore, we split the non-terminals of the grammar between the two players.The non-terminals represent program locations, player *prover* ($\square$) owns the locations where the synthesis should fill in code and player *refuter* ($\bigcirc$) owns the locations where the environment is in control.

Definition 1
A context-free game $\mathcal{G} = (G, A)$ is given by a context-free grammar G with ownership partition of the non-terminals $N = N_{\bigcirc} \uplus N_{\square}$ and a finite automaton A. A play π starting from sentential form α is a left-derivation process starting from α where each player chooses the rule for her non-terminals. We call a play maximal if it represents a derivation process from $\alpha \in \mathscr{S}$ to a terminal word $w \in T^*$. A maximal play π is won by refuter if it ends in a terminal word $w \in \mathcal{L}(\overline{A})$. All other plays are won by prover.

We write $X \in \square$ resp. $X \in \bigcirc$ to denote that X belongs to prover resp. refuter.

The reason why we assign infinite plays to prover is the following. The task of the synthesis algorithm is to ensure that the resulting program satisfies the specification. However an infinite play in the context-free game does not correspond to a valid program and can therefore not dissatisfy the specification. In other words, refuter has to derive a counter-example, i.e. a valid program for which the specification does not hold, in order to win.

We can represent all the plays in the game arena B_G of context-free game $\mathcal{G}$, a graph with ownership partition of the nodes. The nodes represent sentential forms in the plays and the edges the left-derivation relation. The graph is constructed inductively as follows. It has a node labeled by S. For each node labeled by sentential form α, it has an edge to a node labeled with β if $\alpha \Rightarrow \beta$ by a left derivation. Thus, the game arena contains only sentential forms α that can be derived from S. A vertex is owned by the player owning the left-most non-terminal in the sentential form.

Definition 2
The game arena of a context-free game $\mathcal{G}$ is given by $\mathsf{B}_G = (V_{\mathsf{B}_G}, \rightarrow_{\mathsf{B}_G})$, where the set of nodes $V_{\mathsf{B}_G} = V_{\mathsf{B}_G,\square} \uplus V_{\mathsf{B}_G,\bigcirc}$ has an ownership partition for the players. The game arena has the following properties.

- $V_{\mathsf{B}_G} = \{\alpha \in \mathscr{S} \mid S \Rightarrow^* \alpha\}$
- $\rightarrow_{\mathsf{B}_G} = \{(\alpha, \beta) \in V_{\mathsf{B}_G} \times V_{\mathsf{B}_G} \mid \alpha \Rightarrow \beta\}$
- $V_{\mathsf{B}_G,\square} = \{w \in V_{\mathsf{B}_G} \mid w \in T^* \textit{ and } w \in \mathcal{L}(A)\} \cup \{wX\alpha \in V_{\mathsf{B}_G} \mid X \in \square\}$
  $V_{\mathsf{B}_G,\bigcirc} = V_{\mathsf{B}_G,\square} \setminus V_{\mathsf{B}_G}$.

Thus, we can also see a context-free game as a (reachability) **graph game** on the game arena. The positions are given by the sentential forms in the game arena and the moves by the edges. A play in the graph game starting at node v is a path starting at v in the graph. If the position belongs to player $\star \in \{\square, \bigcirc\}$, $\star$ is allowed to choose the next move. The starting position is the node with label S. Refuter wins if the play is maximal, i.e. the last position has no outgoing edge, and the last position belongs to her. As only nodes labeled with terminal words have no outgoing edges and only terminal words $w \notin \mathcal{L}(A)$ belong to refuter, the winning condition matches the one from the grammar game. Any other play, in particular infinite ones, are won by prover.

Note that in general, the game arena is infinite as there are usually infinitely many sentential forms that are derivable from S.

By equipping the set of non-terminals N of $\mathcal{G}_{ex}$ with the ownership partition $N_{\square} = \{S, Y\}$ and $N_{\bigcirc} = \{X\}$, we get the game arena $\mathsf{B}_{\mathcal{G}_{ex}}$.

Note that in the example, the game arena $\mathsf{B}_{\mathcal{G}_{ex}}$ is actually a tree. This is the case because the grammar $\mathcal{G}_{ex}$ is unambiguous. But if the grammar is ambiguous, there exists a sentential

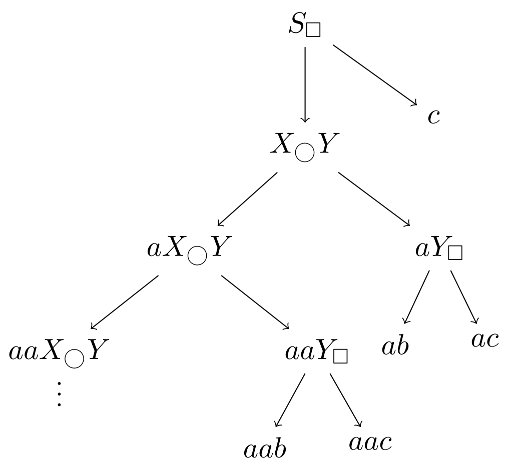


Figure 3.: Game arena $\mathsf{T}_{\mathcal{G}_{ex}}$ induced by grammar $\mathcal{G}_{ex}$. The subscripts indicate the owner of the nodes.

form α s.t. there are at least two derivation processes from S to α. Then, there would be two paths from S to α in the game arena, meaning that it can not be a tree.

To solve the synthesis problem, we have to decide whether prover has a winning strategy for the given context-free game.

Intuitively, a strategy $\mathsf{s}_\star$ tells player $\star$ which grammar rule to apply when it is her turn or in other words to which sentential form in the game graph she should move. In general, it is a function that maps the play up to the current position to a successor (wrt. the edges of the game arena) of the current position. A strategy is winning if the player wins every play where he moves according to the strategy, no matter what the opponent does.

Definition 3

- *A strategy for player $\star \in \{\bigcirc, \Box\}$ is a function $\mathsf{s}_\star : \mathscr{S}^* \mapsto \mathscr{S}$. It maps a sequence of sentential forms $\alpha_1, \ldots, \alpha_n$ to a sentential form α_{n+1}, where $\alpha_n = wX\alpha$ with $X \in \star$ and $\alpha_{n+1} = w\beta\alpha$ with $X \rightarrow_G \beta$.*
- *We call a strategy $\mathsf{s}_\star$ positional if it only depends on the current position of the play, i.e. it can be written as $\mathsf{s}_\star : \mathscr{S} \mapsto \mathscr{S}$.*
- *A play $\pi = \alpha_1, \alpha_2, \alpha_3, \ldots$ is conform to a strategy $\mathsf{s}_\star$, if $\mathsf{s}_\star(\alpha_1, \ldots, \alpha_i) = \alpha_{i+1}$ resp. $\mathsf{s}_\star(\alpha_i) = \alpha_{i+1}$ if the strategy is positional, for all α_i owned by $\star$.*
- *A strategy $\mathsf{s}_\star$ is winning for player $\star$ if all plays that start with S and that are conform to $\mathsf{s}_\star$ are winning for player $\star$.*

We say that player $\star$ can **enforce** the derivation of some set M of sentential forms from α, if player $\star$ has a strategy $\mathsf{s}_\star$ that ensures that each play starting from α and conform to $\mathsf{s}_\star$ eventually reaches a sentential form contained in M.

In the context-free game $\mathscr{G}_{ex} = (\mathcal{G}_{ex}, \mathcal{A}_{ex})$, prover has a winning strategy. In the first move of the play at position S, she has to choose rule $S \rightarrow_G XY$. Then, refuter can derive any number of a's from X. If refuter enforces an infinite play, prover wins by convention. Otherwise, she can move again at position a^nY for some $n \geq 1$. Depending on whether n is even or odd, prover derives b resp. c from Y. Then, the resulting terminal word is contained in $\mathcal{L}(\mathcal{A}_{ex})$ and prover wins. Formally, the winning strategy is given by $\mathsf{s}_\Box(S) = XY$, $\mathsf{s}_\Box(a^{2n}Y) = a^{2n}b$ and $\mathsf{s}_\Box(a^{2n+1}Y) = a^{2n+1}c$ for $n \in \mathbb{N}$.

As the automaton A represents the specification, checking whether refuter has a winning strategy amounts to check whether the program template has an instantiation satisfying the specification. Refuter only wins if she can enforce a valid instantiation which is a counter-example for the inclusion. Prover however also wins if no valid instantiation is derived, i.e. the play is infinite. Thus, refuter actually has the harder task to manage and therefore we consider the context-free games from her point of view. But as context-free games are determined [5], refuter has a winning strategy if and only if prover has none. Thus, an algorithm which checks whether refuter wins the context-free game also solves the synthesis problem.

Part I.

The three methods

3. Summary-based Approach

The idea of the summary-based approach is to represent the game arena by a positive Boolean formula. This formula captures both the derived terminal words and the alternating structure of the game arena, i.e. the influence of the players on the derived word. The alternating structure is represented by the Boolean operators in the formula. The choices of prover are modeled by the operator $\wedge$ and the ones of refuter by $\vee$. The atomic propositions represent the derivable words. As there are potentially infinitely such words, we represent them via summaries, which capture the state changes induced by the words on the automaton A. As there are only finitely many possible state changes, we get a finite set of atomic propositions. In order to derive the winner of the context-free game, we assign truth values to the summaries and thereby evaluate the formula. For a detailed presentation of the summary approach, refer to [5].

The rest of this section is organized as follows. In the first part, we show how we can read off this formula from the game arena and how the formula relates to the game. As the game arena is usually infinite, we can not determine the formula this way. Thus, we will show in the second part how we actually compute the formula, namely by solving a system of equations for the least fixed-point.

3.1. From the game arena to formulas

In order to make this part more readable, we take $\mathscr{G}_{ex} = (\mathcal{G}_{ex}, \mathcal{A}_{ex})$ from Example 2 (with ownership partitioning $N_{\square} = \{S, Y\}$, $N_{\bigcirc} = \{X\}$) resp. Figure 2 as a running example. We use the example to show how we read off the formula.

We handle automaton $\mathcal{A}_{ex}$ as if it was deterministic, although there do not exist transitions for every symbol from every state. If a transition is missing, assume it leads to an error state which has a self-loop for all symbols and is not accepting. For the sake of readability, we omitted this error state. Although we chose a deterministic automaton for the running example, the summary approach also works for non-deterministic automaton. Actually, this is one of the main selling points of this approach. We comment on this below.

Note that prover wins this game. We have that $\left(a^{(2n)}b \mid a^{(2n+1)}c\right) \subseteq \mathcal{L}(\mathcal{A}_{ex})$ for some $n \in \mathbb{N}_0$. To win, prover first chooses rule $S \to XY$. Then refuter can derive an arbitrary number of a's from X. Depending on whether refuter chose an even or an odd number of a's, prover either derives a b or a c from Y. Then the automaton accepts the derived word and prover wins.

Recall the game arena of $\mathscr{G}_{ex}$ from Figure 3. We now show how we can transform this game arena into a formula. The first problem that we have to tackle is that the graph is usually infinite.

This in fact always happens if we have a **loop** in the grammar G i.e. for some non-terminal X there exists a sequence of rules

$$\begin{aligned} &X \to \alpha_1 X_1 \beta_1 \\ &X_1 \to \alpha_2 X_2 \beta_2 \\ &\dots \\ &X_n \to \alpha_n X \beta_n \end{aligned}$$

where X_i are non-terminals and α_i, β_i $(i = 1, \dots, n)$ sentential forms.

Our first observation at this point is that the labels of the inner nodes do not matter. We are only interested in the terminal words that label the leaves. For the inner nodes, we only need to know which of the players owns the positions. Therefore, we will replace the inner nodes by symbols for prover resp. refuter. As we aim for a Boolean formula, we use $\wedge$ for prover and $\vee$ for refuter. We only need conjunction $\wedge$ and disjunction $\vee$ to represent the choices of the players, we do not make use of the negation $\neg$. Thus, we will use positive Boolean formulas to capture the game arena.

Definition 4
A positive Boolean formula F over domain $\mathcal{D}_F$ is inductively defined by

$$F := p \mid G \vee G' \mid G \wedge G'$$

where G, G' are formulas and $p \in \mathcal{D}_F$ an atomic proposition.

Note that we left the set of atomic propositions $\mathcal{D}_F$ unspecified for now. Throughout this subsection, we will consider positive Boolean formulas with different atomic propositions and always indicate which set we use as domain. At the end of the subsection, we fix a set of atomic propositions.

By applying the first observation to our arena, we get the tree depicted in Figure 4.

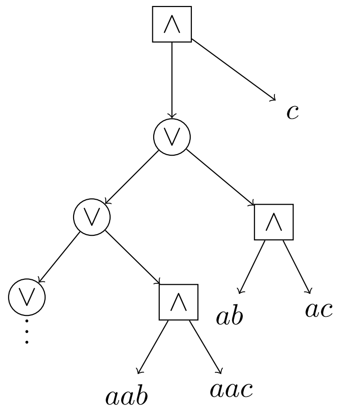


Figure 4.: Modified tree of plays.

Now, we can view the game arena as a syntax tree of a Boolean formula with atomic propositions $\mathcal{D}_F = T^*$. We can read off the formula by a top-down traversal of the tree. For the running example, we get the following formula:

$$c \wedge ((ab \wedge ac) \vee (aab \wedge aac) \vee (aaab \wedge aaac) \vee \dots)$$

This formula captures all the plays. At the beginning of the plays, prover can either choose to derive a c or she can move to position XY. In the formula, this is represented by the outermost $\wedge$. The left clause containing only c stands for the first case, the second clause for the other. Now it is refuters turn. She can derive an arbitrary number of a's from X and finally, prover ends the game by either appending an b or a c. Therefore, the formula contains the clauses $(a^n b \wedge a^n c)$ for $n \geq 1$, separated by a $\vee$.

This formula is again infinite. Even worse, also the set of atomic propositions $\mathcal{D}_F = T^*$ is infinite. To tackle this problem, we observe that the actual terminal words are not important. We only need to know whether automaton A accepts the derived word or not. For this task, it is sufficient to know the state changes that our derived word w induces on the automaton. By state changes we mean the set $\{(q,p) \mid q,p \in Q\}$ where p is s.t. $q \xrightarrow{w} p$. We consider two words w and v as equivalent if they induce the same state changes in A. The idea of state changes and equivalence is formalized using the **transition monoid** of A.

3.1.1. The transition monoid

To obtain a finite set of atomic propositions, we introduce the equivalence relation $\sim_A$ factorizing the set of atomic propositions T^* into a finite number of equivalence classes.

Definition 5
Let A be a finite automaton. By $\sim_A$, we denote the transition equivalence relation on words $w_1, w_2 \in T^*$, given by

$$w_1 \sim_A w_2 \iff \left(\forall q, q' \in Q : \; q \xrightarrow{w_1} q' \iff q \xrightarrow{w_2} q'\right).$$

Thus, all the words contained in the same equivalence class induce the same state changes on A. The equivalence classes can be seen as the elements of the transition monoid of A.

Definition 6
Let A be a finite automaton. Then the transition monoid of A is given by

$$M(A) = (B(A) = \mathcal{P}(Q \times Q), ;, id)$$

The elements of the transition monoid are relations over states of A. We call them boxes and denote them by τ, ρ. The relational composition ; is defined by

$$\rho; \tau = \{(q, p) \mid \exists q' \in Q : (q, q') \in \rho \text{ and } (q', p) \in \tau\}.$$

The element $id = \{(q, q) \mid q \in Q\}$ is the identity element of the monoid, i.e.

$$id; \rho = \rho; id = \rho.$$

We explained that the elements of the monoid correspond to the equivalence classes induced by $\sim_A$. The relations represented by a box are exactly the state changes induced by the words in the equivalence class. We call the set of words contained in an equivalence class the language of the box corresponding to this class.

Definition 7
The language of box ρ is given by

$$\mathcal{L}(\rho) = \{w \in T^* \mid (q, p) \in \rho \iff q \xrightarrow{w}{}^{*} p\}.$$

If $w \in \mathcal{L}(\rho)$, we say that ρ is the box of w. In general, we denote the box of a word w by ρ_w.

The name“box” for the elements of $M(A)$ comes from their graphical representation as boxes. Figure 5 pictures some of the boxes for our running example.

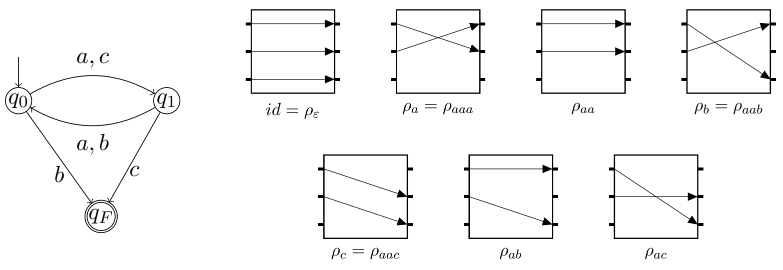
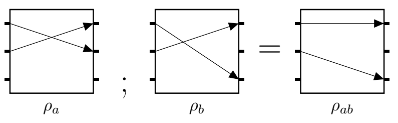


Figure 5.: Subset of the boxes of our running example. The upper dash on each side of the boxes represents state q_0, the middle dash q_1 and the lower dash q_F. The arrows between the dashes mark the state changes. We only pictured the relevant boxes, those of words that are actually derivable from $\mathcal{G}_{ex}$. The boxes of $\rho_{bb}, \rho_{cc}, \rho_{bc}, \rho_{cb}$ are left out.

Note that our automaton $\mathcal{A}_{ex}$ is deterministic. Therefore, we only have (at most) one arrow originating from each state. In case of a non-deterministic automaton, we would have several arrows originating from a single dash. The methods presented below still apply and are not influenced by the non-determinism.

We can compute the boxes for words $w = w_1 \dots w_n$ from the boxes of their individual letters w_i. To this end, we use the relational composition operator ; by $\rho_w = \rho_{w_1}; \rho_{w_2}; \dots; \rho_{w_n}$.

In terms of boxes, we can derive the box of the composed word by connecting the arrows of ρ_{w_i} with the ones of $\rho_{w_{i+1}}$. In our running example, we can get the box of ρ_{ab} as follows.



The composition operator is **associative**, e.g.

$$(\rho_a; \rho_b); \rho_c = \rho_a; (\rho_b; \rho_c) = \rho_a; \rho_b; \rho_c.$$

This is an important property, as it allows us to compose multiple boxes at once instead of gradually composing from left to right.

3.1.2. Back to the game arena

By Figure 5, we get that $\mathcal{L}(\mathcal{A}_{ex})$ can be factorized into eleven equivalence classes/boxes . By replacing each word w by its box ρ_w in the leafs of the game arena, we get a finite set of atomic propositions $\mathcal{D}_F = B(A)$ as desired. For the rest of the thesis, we fix the atomic propositions $\mathcal{D}_F$ of our formulas to be the set of boxes $B(A)$. Figure 6 pictures the tree resulting from this replacement.

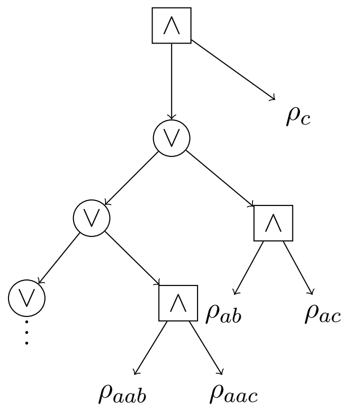


Figure 6.: Tree of plays resulting from replacing the terminals by their boxes.

Top-down traversal gives us the corresponding formula.

$$\rho_c \wedge ((\rho_{ab} \wedge \rho_{ac}) \vee (\rho_{aab} \wedge \rho_{aac}) \vee (\rho_{aaab} \wedge \rho_{aaac}) \vee \dots)$$

The formulas may still be infinite (and in our case is), although the set of atomic propositions is now finite. To solve this problem, note that up to logical equivalence these infinite formulas represent functions $2^{M(A)} \to \{0, 1\}$. But all such functions can also be represented by finite formulas. For each of the infinite formulas, we can find a logically equivalent finite formula. For our running example, we can use that $\rho_{ab} = \rho_{a^{(2n+1)}b}$, $\rho_b = \rho_{a^{(2n)}b}$ and $\rho_{ac} = \rho_{a^{(2n+1)}c}$, $\rho_c = \rho_{a^{(2n)}c}$. Then, get the following equivalent formula

$$\begin{aligned} &\rho_b \wedge ((\rho_{ab} \wedge \rho_{ac}) \vee (\rho_{aab} \wedge \rho_{aac}) \vee (\rho_{aaab} \wedge \rho_{aaac}) \vee \dots) \\ \Leftrightarrow &\rho_b \wedge ((\rho_{ab} \wedge \rho_{ac}) \vee (\rho_{aab} \wedge \rho_{aac})). \end{aligned}$$

It is not surprising that we can represent the whole tree of plays by a finite formula. Intuitively, plays in an infinite branch have to be repetitive in terms of alternating structure and boxes of the derived words, as the infinite branches stem from a loop in the grammar. Thus, after reaching a certain depth in the game arena, no new plays will appear.

In $\mathscr{G}_{ex}$ for example, the (unique) plays ending in ab resp. $aaab$ bear strong similarities. First, the boxes of both words are the same. Second, the plays are very similar in their alternating structure. In the beginning of the plays, prover had to choose rule $S \to XY$, then refuter had to derive the appropriate number of a's (one or three) and finally prover had to derive a b from Y. Thus, we can represent both plays by box ρ_{ab} in the clause $(\rho_{ab} \wedge \rho_{ac})$.

Until now, we only took into account the possible plays. To derive the winner of the game, we assign truth values to the boxes. We are interested in the state changes starting from q_0 of the boxes.

Definition 8
We call a box rejecting if it does not contain any relation (q_0, q_F) where q_0 is the starting state and $q_F \in Q_F$ is a final state. If the box does contains such a relation, we call it accepting.

As we play from the point of view of refuter, we assign value *true* to all rejecting boxes and *false* to all others. If the formula evaluated to *true*, refuter wins the game, otherwise prover does.

In our running example, we assign true to $\rho_b, \rho_{ab}, \rho_{aac}$ and false to $\rho_c, \rho_{ac}, \rho_{aab}$. Then the formula evaluates to *false* which confirms prover as the winner of $\mathscr{G}_{ex}$.

3.2. Representation of infinite plays

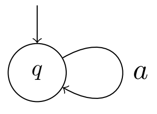
Before we show how to compute the formulas, we need to clarify how we represent infinite plays. In our running example, the game arena contains the infinite play

$$S \Rightarrow XY \Rightarrow aY \Rightarrow aaY \Rightarrow aaaY \Rightarrow \dots.$$

As explained in Section 2, an infinite play is always winning for prover as refuter only wins if she derives a counter-example to the inclusion in $\mathcal{L}(A)$. In $\mathscr{G}_{ex}$, it is technically possible for refuter to enforce an infinite play. But as we would loose this play, we can safely ignore this possibility and not capture this play in our formula.

Consider the following context-free game, given by $\mathcal{G}'_{ex}$ and $\mathcal{A}'_{ex}$: Prover can either derive

$$S_\Box \to a \mid aS$$



any word $w \in a^+$ or enforce an infinite play. The alphabet $\Sigma' = \{a\}$ only contains the terminal a. The automaton $\mathcal{A}'_{ex}$ does not have a final state and thus does not accept any word. The transition monoid of automaton $\mathcal{A}'_{ex}$ only contains the boxes *id* and $\rho_a = \rho_{a^n}$ $(n \geq 1)$, which are both rejecting.

If we represent the game arena by a formula as above, we get the formula which only consists of the atomic proposition ρ_a. But evaluating this formula would lead to value *true*, suggesting that refuter wins this game.

This means that we cannot always ignore the infinite plays. Until now, we represented the game arena by a formula from the set of positive Boolean formulas over $B(A)$. Our approach is to augment this set by the unsatisfiable formula *false* and thereby represent infinite plays.

Definition 9
The set of formulas BF_A used to represent the game arena is given by the positive Boolean formulas over $B(A)$ joined with the unsatisfiable formula false. We evaluate the formula false on the syntactic level.

$$F \vee \mathit{false} = \mathit{false} \vee F = F \qquad\qquad F \wedge \mathit{false} = \mathit{false} \wedge F = \mathit{false}$$

We evaluate *false* on the syntactic level to facilitate the definition of the relational composition (see below). By these evaluation rules, *false* will spread trough the formula if prover can enforce an infinite play, resulting in the unsatisfiable formula $false$.

For our running example $\mathscr{G}_{ex}$, the addition of $false$ to the formula does not make a difference, as expected.

$$\begin{aligned}
&\rho_b \wedge (false \vee (\rho_{ab} \wedge \rho_{ac}) \vee (\rho_{aab} \wedge \rho_{aac}) \vee (\rho_{aaab} \wedge \rho_{aaac}) \vee \dots) \\
=\; &\rho_b \wedge ((\rho_{ab} \wedge \rho_{ac}) \vee (\rho_{aab} \wedge \rho_{aac}) \vee (\rho_{aaab} \wedge \rho_{aaac}) \vee \dots) \\
\Leftrightarrow\; &\rho_b \wedge ((\rho_{ab} \wedge \rho_{ac}) \vee (\rho_{aab} \wedge \rho_{aac})).
\end{aligned}$$

But for $\mathscr{G}'_{ex}$, the change leads to the following formula

$$\rho_a \wedge false = false$$

which is unsatisfiable as desired.

We will see in the next section that $false$ will naturally arise from infinite plays while computing the formulas.

This concludes the section about the intuition behind the formula and how it relates to the tree of plays. It remains to show how we actually compute the formulas.

3.3. Computing the formulas

The task is to capture all the possible plays of the context-free game by a finite Boolean formula or the unsatisfiable formula *false*. This includes both the derivable terminal words and also the influence that the players have on the derivation. As explained in the previous section, the terminal words (or rather their boxes) are represented by the atomic propositions and the operators capture the choices of the players.

We emphasize the two-step approach of our method. First, we compute a formula which only captures all possible plays, regardless whether they are winning for refuter or not. The winner of the game is only determined in the second step, where we assign truth values to the atomic propositions (the boxes) and evaluate the formula.

The key insight is that we can derive all possible plays from some sentential form XY on from all possible plays starting from X resp. Y. Intuitively, a maximal play from XY is a play from X to some terminal word v, followed by a play from Y to some word w, with v prepended. This means that we can consider the plays from X resp. Y on their own and compose them with each other afterwards instead of appending the plays starting from Y to all the plays that started from X. Figure 7 depicts the difference between the two views.

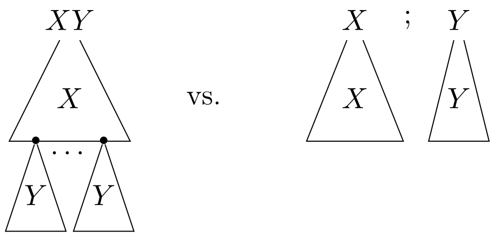


Figure 7.: The two views on plays from XY. We denote the composition operator by ;.

As we use formulas to represent the plays, we show how the composition of plays works on formulas. Suppose therefore that we have the formulas G resp. F for the game arena starting at X resp. Y. We essentially lift the composition operator ; of the transition monoid from boxes to formulas over boxes. We compose every box from formula G with each box from F while preserving the alternating structure of both formulas.

$$\begin{tikzcd}
 & & S \arrow[dl] \arrow[dr] & \\
 & X \arrow[dl] \arrow[dr] & & Y \arrow[d] \\
a & & X \arrow[d] & c \\
 & & a &
\end{tikzcd}$$

Figure 8.: Parse tree of a play/derivation process from S to aac. At each non-terminal, a grammar rule is applied. Thus, on each path ≤ 3 rules are applied.

Definition 10
The composition over BF_A is defined by $F; false = false; F = false$ and for composite formulas we have

$$(F_1 \bigotimes F_2); G = F_1; G \bigotimes F_2; G \qquad\qquad \rho; (G_1 \bigotimes G_2) = \rho; G_1 \bigotimes \rho; G_2$$

where F, F_1, F_2, G_1, G_2 are positive Boolean formulas and $\bigotimes \in \{\vee, \wedge\}$.

Using this insight and the fact that we are only interested in the possible plays (and not yet who wins them), we can also investigate the parse tree instead of the tree of plays.

A **parse tree** of a derivation process in grammar G is a tree rooted at a node labeled by S. Each inner node is labeled by a non-terminal of G and each leaf by a terminal. If a node X in the tree has children $\alpha_1, \ldots, \alpha_n$ ($\alpha_i \in T \cup N$), then there exists rule $X \rightarrow_G \alpha_1 \ldots \alpha_n$ in G. Thus, we apply a rule at each inner node.

A parse tree only represents a single derivation process. Consider for example the parse tree of the derivation process from S to aac in $\mathcal{G}_{ex}$ depicted in Figure 8. But as we consider a context-free game where both players influence the derivation process, we cannot focus on a particular process. Therefore, we modify the definition of the parse tree, such that it is able to represent all parse trees at once i.e. all possible derivations in the grammar.

However, we need to make sure that the right-hand sides of different rules do not get mixed up. To solve this issue, we introduce intermediary nodes that represent the composition of the symbols on the right-hand side of a rule. From a node with label X, we have for each rule of shape $X \rightarrow_G \alpha_1 \ldots \alpha_n$ an edge to a newly introduced node with label $\alpha_1; \ldots; \alpha_n$. The children of this node are labeled by α_i ($i = 1, \ldots, n$) as in the regular parse tree. Furthermore, we replace the terminal symbols at the leafs by the box of the symbol.

Definition 11
The augmented parse tree represents all possible plays/derivation processes in a context-free game. The augmented parse tree has the following properties

- *The root is labeled by S,*
- *the inner nodes are either labeled by a non-terminal X or by a composition $\alpha_1; \ldots; \alpha_n$ of symbols $\alpha_i \in T \cup N$,*
- *the roots are labeled by boxes ρ_a of terminals $a \in T$,*
- *there is an edge $X \rightarrow \alpha_1; \ldots; \alpha_n$, if there exists a grammar rule $X \rightarrow_G \alpha_1 \ldots \alpha_n$,*
- *there is an edge $\alpha_1; \ldots; \alpha_n \rightarrow \alpha_i$, if $\alpha_i \in N$ is a non-terminal and*
- *there is an edge $\alpha_1; \ldots; \alpha_n \rightarrow \rho_{\alpha_i}$, if $\alpha_i \in T$ is a terminal.*

Note that there may be several nodes with the same label.

As we are dealing with context-free games, there are two types of nodes labeled with a non-terminal. If the node is labeled with non-terminal X belonging to refuter, it is called $\vee$-node and $\wedge$-node otherwise.

If we apply this transformation to our running example, we get the tree on the right in Figure 9. The $\vee$-nodes are depicted by a circle and the $\wedge$-nodes by a square. On the left, we show the tree of plays (with boxes instead of terminals) to emphasize the relation of both trees as well as the differences.

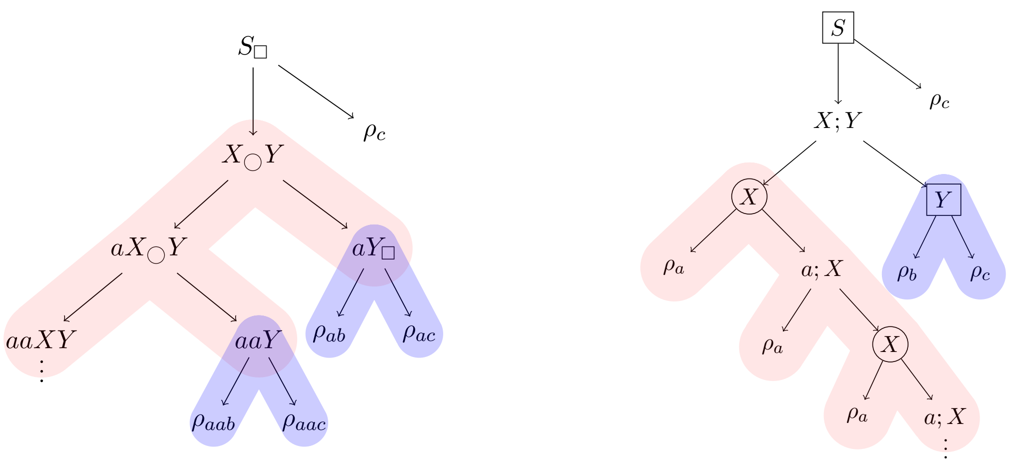


Figure 9.: Game arena (left) in comparison to the extended parse tree (right).

The possible derivation processes starting from X are colored in red and the ones starting from Y in blue. In the game arena, we see that the plays from Y on are appended to the plays starting at X. In the parse tree, the plays are executed in parallel and joined by the composition operator.

We could now read off the formula from the parse tree in a similar fashion than for the tree of plays. We view the $\vee$-nodes resp. $\wedge$-nodes as the operator $\vee$ resp. $\wedge$ and the (new) intermediary nodes as the composition operator. For our running example, this leads to the following formula, which is as expected.

$$\begin{aligned}
&\rho_c \wedge (\rho_a \vee \rho_a; (\rho_a \vee \rho_a; (\dots))); (\rho_b \wedge \rho_c) \\
\Leftrightarrow &\rho_c \wedge (\rho_a \vee \rho_{aa}); (\rho_b \wedge \rho_c) \\
\Leftrightarrow &\rho_c \wedge ((\rho_{ab} \wedge \rho_{ac}) \vee (\rho_{aab} \wedge \rho_{aac}))
\end{aligned}$$

But this method is still impracticable as the parse tree contains infinite paths if the tree of plays does.

Our next observation is that in a parse tree all subtrees whose roots have the same label are equal. In the parse tree of $\mathcal{G}_{ex}$ for example, all subtrees rooted at a node with label X are completely similar. As the infinite paths are caused by loops in the grammar, we replace the infinite paths by appropriate loops in the parse tree. For our running example, we get the (finite) graph depicted in Figure 10.

Formally, this graph is defined as follows for any context-free game given by context-free grammar G and finite automaton A. We call the graph circuit graph.

Definition 12
The circuit graph is given by $(V_\square \cup V_\bigcirc \cup V_{gate} \cup V_{box}, E_{rule} \cup E_{rhs})$ *where*

- $V_\square = \{X \mid X \in N_\square\}$

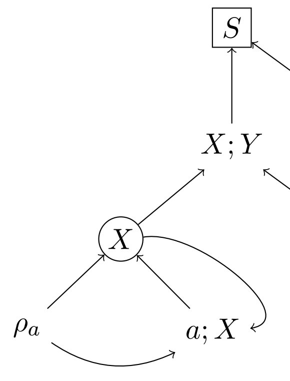


Figure 10.: Circuit graph of the running example.

- $V_{\bigcirc} = \{X \mid X \in N_{\bigcirc}\}$
- $V_{gate} = \{\alpha_1; \dots; \alpha_n \mid \alpha_1 \dots \alpha_n \textit{ is the rhs of some rule in } G \textit{ with } n \geq 2\}$
- $V_{box} = \{\rho_a \mid a \in T\}$
- $$\begin{aligned} E_{rule} = \; & \{(\alpha_1; \dots; \alpha_n, X) \mid X \rightarrow_G \alpha_1 \dots \alpha_n \textit{ is a rule in } G \textit{ with } n \geq 2\} \\ & \cup \{(\alpha, X) \mid X \rightarrow_G \alpha \textit{ is a rule in } G \textit{ with } \alpha \in N\} \\ & \cup \{(\rho_\alpha, X) \mid X \rightarrow_G \alpha \textit{ is a rule in } G \textit{ with } \alpha \in T\} \end{aligned}$$
- $$\begin{aligned} E_{rhs} = \; & \{(\alpha_i, \alpha_1; \dots; \alpha_n) \mid \alpha_i \in V_{\Box} \cup V_{\bigcirc} \textit{ and } \alpha_1; \dots; \alpha_n \in V_{gates}\} \\ & \cup \{(\rho_{\alpha_i}, \alpha_1; \dots; \alpha_n) \mid \rho_{\alpha_i} \in V_{box} \textit{ and } \alpha_1; \dots; \alpha_n \in V_{gates}\}. \end{aligned}$$

The nodes in $V_{\Box}$ resp. $V_{\bigcirc}$ represent the non-terminals owned by the respective players and the nodes in V_{box} the (boxes of) the terminal words. The nodes in V_{gate} stand for the right-hand sides of the grammar rules. The interpretation of the edges is the same as for the augmented parse tree.

The circuit graph is actually very similar to the augmented parse tree. The main difference is that the circuit graph contains at most one node for each possible label. In the augmented parse tree this was not the case. There may be several nodes with the same label. Suppose we build the augmented parse tree top-down starting from the root and want to add an edge $l_1 \rightarrow l_2$ (l_1, l_2 are labels) from some already existing node v_1 with label l_1 in the parse tree. Then, we always add a new node v_2 with label l_2 to the tree, no matter whether a node with the same label is already contained in the tree. For the circuit graph, we reuse the node with label l_2 if such a node already exists in the tree. Thus, the resulting graph has one vertex for each non-terminal, one for each (box of) a terminal and one for each right-hand side of a rule and is thereby finite.

In the infinite trees, we could traverse the infinite paths top-down until no more "new" plays appeared. As we replaced infinite paths by loops, we instead need to iterate the loop until no new plays are found. Thus, we employ a fixed-point iteration on the graph to get the formulas. We interpret the graph as a circuit. The nodes of $V_{\Box}$ and $V_{\bigcirc}$ are used to store intermediate values and are initialized with *false*. The nodes in $V_{\Box}$ combine their input values by an $\wedge$, the ones from $V_{\bigcirc}$ by a $\vee$ and store the resulting formula. The nodes of V_{gates} are viewed as gates, that return the composition of their input. The nodes of V_{box} always store the value that they are labeled with and are not updated. They deliver the input to the circuit.

This interpretation as a circuit also explains why we flipped the direction of the edges. In the extended parse tree, the nodes labeled by X represent this exact same non-terminal X and its children are labeled by the right-hand sides γ of the grammar rules $X \rightarrow_G \gamma$. The formulas are read off the tree by a top-down traversal. Moving downwards in the tree from X to one of its children γ corresponds to applying grammar rule $X \rightarrow_G \gamma$. In the circuit, we view the nodes labeled by X as a variable. Variable X holds the information about the plays starting from X. Such a play starts with a rule $X \rightarrow_G \gamma$ and continues as a play from γ. The plays from γ can be computed from the composition of the plays from $\gamma_1, \ldots, \gamma_n$. The composition is executed by the gate $\gamma_1; \ldots; \gamma_n$ succeeding node X. The nodes $\gamma_1, \ldots, \gamma_n$ send their current value trough the composition gate to node X, which combines all his inputs by either $\vee$ or $\wedge$ and stores the resulting value. In every iteration, each node sends its value along the outgoing edges and stores the input from the incoming edges until a fixed-point is reached. The recalculation of the value of variable X corresponds to the application of all rules with left-hand side X. Note that the order in which the values are recalculated does not matter up to logical equivalence.

By initializing the value of each node labeled by a non-terminal X by *false*, we handle the plays starting from X as infinite at first. If prover can enforce an infinite play from X, the value *false* is never replaced. Otherwise, the value is replaced by a formula at some point.

In the running example, refuter can enforce the finite play $\pi = X, a$ from X and indeed the value of node X is replaced by $\rho_a \vee \rho_a; false = \rho_a$ in the first iteration.

To be precise, in the k-th iteration, the value of X captures all plays where at most k rules are applied in each branch of the corresponding parse tree (at each inner node a rule is applied). For example, in the parse tree of the play from S to aac, less than 3 rules are applied in each of its branches. (Figure 8).

Inductively, we can verify this claim as follows. Before the first iteration, the values of all nodes labeled with a non-terminal X are equal to *false* and the nodes labeled by a box ρ_a store the value coinciding with their label. Indeed, only plays starting from a terminal have at most 0 rules applied in each branch of their parse tree. Suppose now that this property holds in the k-th iteration for all variables. In the $k+1$-th iteration, variable X stores the composition of the values of variables $\gamma_1, \ldots, \gamma_n$ if there exists a grammar rule $X \rightarrow_G \gamma_1, \ldots, \gamma_n$. In other words, X captures all plays π that can be assembled from plays π_i stored in the variables γ_i. Thus, for all plays π from X, there exists a parse tree where at most $k+1$ rules are applied in each branch. This tree can be assembled from the parse trees of the plays π_i that compose π. The resulting parse tree has root X and we attach the parse trees of the composing plays π_i. From the induction hypothesis we know that they have have at most k rules applied in each branch. Then, as we apply rule $X \rightarrow_G \gamma_1, \ldots, \gamma_n$ at the root, it follows that in the parse tree of π has at most $k+1$ rules are applied in each branch. In Figure 8 for example, the parse tree for the play from S to aac can be assembled from the parse trees of the plays from X to aa and from Y to c which themselves have at most 2 rules applied in each branch.

Instead of conducting the fixed-point iteration on the circuit, we translate it into a system of equations. As the circuit closely reflects the rules of the grammar, we actually do not need to take the detour through the parse tree and the circuit. Instead, we can directly derive the system from the formulas.

Definition 13
The system of equations induced by the grammar G and the automaton A has the following properties.

- *We have a variable Δ_X for each non-terminal $X \in N$.*
- *We have a variable Δ_a for each terminal $a \in T$ which is defined by $\Delta_a := \rho_a$. We regard the value of the variable as fixed, as it is not altered during the fixed-point iteration.*
- *For each non-terminal X, we have one defining equation.*

$$\Delta_X := \bigotimes_{X \rightarrow \alpha_1 \ldots \alpha_n} \Delta_{\alpha_1}; \ldots ; \Delta_{\alpha_n}, \text{ where } \bigotimes = \begin{cases} \vee & \text{if } X \in \bigcirc \\ \wedge & \text{if } X \in \square \end{cases}$$

For our running example, we have the following system of equations.

$$\begin{aligned}
\Delta_S &= \Delta_c \wedge \Delta_X;\Delta_Y = \rho_b \wedge \Delta_X;\Delta_Y \\
\Delta_X &= \Delta_a \vee \Delta_a;\Delta_X = \rho_a \vee \rho_a;\Delta_X \\
\Delta_Y &= \Delta_b \wedge \Delta_c = \rho_b \wedge \rho_c
\end{aligned}$$

The correspondence to the circuit graph depicted in Figure 10 is immediate.

As we want to use formulas in a fixed-point iteration, we need to define a partial ordering on them. Because we take refuters point of view in the game, a formula F should be bigger than another F' if F' makes it easier for refuter to win. Deciding whether refuter wins means evaluating the formula by assigning *true* to all rejecting boxes and *false* to all accepting ones. Therefore F' is bigger than F if $F \Rightarrow F'$. Unfortunately, implication $\Rightarrow$ is not antisymmetric on our domain BF_A. Therefore, we take BF_A modulo logical equivalence instead. We can understand each formula as a representative of its equivalence class and extend $\Rightarrow$ to the new domain $\mathsf{BF}_A/_{\Leftrightarrow}$ by applying it to representatives of the equivalence classes. The operations $\vee$, $\wedge$ and ; are also lifted to the new domain by applying them to arbitrary representatives. The well-definedness of the operations on the new domain follows from the transitivity of the implication $\Rightarrow$ and the monotonicity of the operations wrt. $\Rightarrow$. The Boolean operators are clearly monotonic, for the proof that the composition operator ; is monotonic, refer to Lemma 6 in [5]. The least element of the partial ordering is $false$. This gives us the desired partial ordering.

We aim for the least solution of our system of equations wrt. component wise $\Rightarrow$. Due to the monotonicity of the operations, we can understand the right-hand side of each equation as a monotonic function $f_X : (\mathsf{BF}_A/_{\Leftrightarrow})^N \mapsto \mathsf{BF}_A/_{\Leftrightarrow}$. The argument of f_X is a vector of formulas, one for each non-terminal, and the output is obtained by plugging the values of the argument into the right-hand side of the equation for X. Combining the functions for each non-terminal gives us the monotonic update function $f : (\mathsf{BF}_A/_{\Leftrightarrow})^N \mapsto (\mathsf{BF}_A/_{\Leftrightarrow})^N$. Thus, according to [3], there exists a unique least solution for the equation $f(\Delta) = \Delta$ and we can compute the solution by a Kleene-iteration:

$$\perp, f(\perp), f(f(\perp)), f(f(f(\perp)))\dots$$

where $\perp = (\mathit{false})^N$ is the N-dimensional vector containing *false* in every component.

As $\mathsf{BF}_A/_{\Leftrightarrow}$ with partial order $\Rightarrow$ is a finite bottomed partial order, this sequence stabilizes into the desired fixed-point solution of f.

We denote the intermediate solutions of the i-th iteration by $\sigma_X^{(i)}$ for each non-terminal and the least fixed-point solution will be denoted by σ_X. The solutions $\sigma_X^{(k)}$ actually correspond to the values of the variable X in the corresponding circuit in the k-th iteration. Thus, $\sigma_X^{(k)}$ captures the plays starting from X that have at most k rules applied in each branch of their parse tree.

For our running example, the Kleene-iteration produces the following solutions.

i	$\sigma_S^{(i)}$	$\sigma_X^{(i)}$	$\sigma_X^{(i)}$
0	$false$	$false$	$false$
1	$false$	ρ_a	$\rho_b \wedge \rho_c$
2	$\rho_c \wedge \rho_{ab} \wedge \rho_{ac}$	$\rho_a \vee \rho_{aa}$	$\rho_b \wedge \rho_c$
3	$\rho_c \wedge ((\rho_a \vee \rho_{aa});(\rho_b \wedge \rho_c))$	$\rho_a \vee \rho_{aa} \vee \rho_{aaa}$	$\rho_b \wedge \rho_c$
	$\Leftrightarrow \rho_c \wedge ((\rho_{ab} \wedge \rho_{ac}) \vee (\rho_{aab} \wedge \rho_{aac}))$	$\Leftrightarrow \rho_a \vee \rho_{aa}$	
4	$\rho_c \wedge ((\rho_{ab} \wedge \rho_{ac}) \vee (\rho_{aab} \wedge \rho_{aac}))$	$\rho_a \vee \rho_{aa}$	$\rho_b \wedge \rho_c$

As expected, we get a formula $\sigma_S^{(4)} = \sigma_S = \rho_c \wedge ((\rho_{ab} \wedge \rho_{ac}) \vee (\rho_{aab} \wedge \rho_{aac}))$ that is equivalent (in our case even equal) to the one that we read off the game arena.

The correctness of the summary approach is proven in [5] (Theorem 22 and 29). The proof determines winning strategies for prover resp. refuter from the solutions $\sigma_X^{(k)}$ in conjunctive normal form.

3.4. Disjunctive Normal Form

In the previous sections, we discussed that the formulas σ_X capture both the derived terminal words and the influence of the players on the derivation processes starting from non-terminal X. It is actually possible to flatten down the choices of the players in the game to two successive choices, one per player. After each player made her choice, the (box of the) derived terminal word is determined. Therefore, we transform the formula σ_X into **disjunctive normal form** (**DNF**). A formula is in DNF if it is of shape

$$\bigvee_{i}^{n} K_i, \text{ where } K_i = \bigwedge_{j}^{m} \rho_{ij}$$

for atomic propositions $\rho_{ij} \in B(A)$. We call the subformulas K_i the clauses of the formula.

Let us consider the formula capturing the plays from S in $\mathscr{G}_{ex}$ again.

$$\rho_c \wedge ((\rho_{ab} \wedge \rho_{ac}) \vee (\rho_{aab} \wedge \rho_{aac}))$$

The green $\wedge$ in the formula represents the choice of prover whether to apply rule $S \rightarrow_G c$ or rule $S \rightarrow_G XY$. The red $\vee$ captures refuters choice whether to derive an even or an odd number of a's. Finally, the orange $\wedge$ represents again provers choice, where he can choose whether to derive terminal b or c from Y.

If we transform the formula into DNF, we have to multiply out the operators. This naturally leads to the flattened decisions of the players. The choice of refuter corresponds to picking a clause from the formula. She is able to enforce the derivation of some terminal word w s.t. ρ_w is contained in the clause. However, in general she cannot choose a particular box from the clause. This decision belongs to prover who is allowed to pick a box from the chosen clause. The choice of prover then fixes the derived terminal word. By transforming the formula above into DNF, we get the following formula.

$$(\rho_c \wedge \rho_{ab} \wedge \rho_{ac}) \vee (\rho_c \wedge \rho_{aab} \wedge \rho_{aac})$$

The left clause corresponds to refuter deriving an odd number of a's from X and the right clause to the derivation of an even number provided that prover picks rule $S \rightarrow_G XY$ in the first turn of the play. Suppose refuter picked the left clause. If prover chooses the first atomic proposition, this compares to prover picking rule $S \rightarrow_G c$ in the first turn. If she picks one of the other propositions, say ρ_{ab}, this is conform to her first deriving XY from S and then appending b to the odd number of a's derived by refuter.

We will see that the other approaches to solve context-free games use a similar flattening of the choices of the players.

3.5. Overview of the summarization method

We conclude the section by a brief overview of the summarization method. Let $\mathscr{G}$ be a context-free game given by a context-free grammar and a finite automaton. To decide the winner of $\mathscr{G}$, we employ the following steps.

1. Compute the box of each terminal $a \in T$ from A.
2. Set up the system of equations as described in Definition 13.

3. Employ a Kleene iteration until the least fixed-point of the system of equations is reached.

4. Evaluate the formula σ_S of the starting symbol S by assigning *true* to all rejecting boxes and *false* to all accepting boxes. If $\sigma_S = true$, refuter wins, otherwise prover does.

4. Saturation-based Approach

The idea of the saturation-based approach is to saturate an automaton by adding transitions to augment the accepted language. In our setting, we start by an automaton $\mathscr{A}$ which accepts $\overline{\mathcal{L}(A)} = T^* \setminus \mathcal{L}(A)$ (recall that we play from refuters point of view). The goal of the saturation is to construct an automaton by adding transitions s.t. also accept the sentential forms contained in the attractor $\mathsf{Attr}_{\bigcirc} = \mathsf{Attr}_{\bigcirc}(\overline{\mathcal{L}(A)}, \Rightarrow)$ are accepted.

Informally, the idea behind the attractor $attr_\star(M, r)$ for a starting set $M \subseteq \mathscr{S}$ and a transitive relation $r \subseteq \mathscr{S} \times \mathscr{S}$ is the following. The attractor iteratively collects sentential forms α from which player $\star$ can enforce a path $\alpha = \alpha_0, \alpha_1, \ldots, \alpha_n = \beta$ from α to sentential form $\beta \in M$ where $(\alpha_i, \alpha_{i+1}) \in r$ for $i = 0, \ldots, n$. If r is the left-derivation relation for example, a path corresponds to a derivation process. We start with set $M_0 = M$ in the 0-iteration, as player $\star$ can enforce a path of length 0 to a sentential form contained in M from any $\alpha \in M$. Consider now any sentential form $\alpha \in \mathscr{S}$. If α belongs to $\star$ and there is a tuple $(\alpha, \beta) \in r$ for $\beta \in M$. Then player $\star$ can enforce a path from α to a sentential form in M by using exactly this tuple. If α belongs to the other player, $\star$ is not allowed to choose the next position and can only enforce a visit to M if it holds for all edges $(\alpha, \beta) \in r$ that $\beta \in M$. We collect all the sentential forms α for which one of the above cases holds in set M_1. Note that in particular $M_0 \subseteq M$. In the next iteration, we collect sentential forms in the same manner, but this time with M_1 as base instead on $M_0 = M$. In iteration $i + 1$, we use set M_i as base to construct M_{i+1}.

Definition 14
Formally, the attractor $\mathsf{Attr}_{\bigcirc}(M, r)$ *for player refuter, for a starting set* $M \subseteq \mathscr{S}$ *and a transitive relation* $r \subseteq \mathscr{S} \times \mathscr{S}$ *is defined as follows.*

$$\begin{aligned}
\mathsf{Attr}_{\bigcirc}(M,r)^0 &= M \\
\mathsf{Attr}_{\bigcirc}(M,r)^{i+1} &= \mathsf{Attr}_{\bigcirc}(M,r)^i \\
&\qquad \cup \{wX\alpha \in \mathscr{S}^* \mid X \in \bigcirc \textit{ and } \exists\ (wX\alpha, \beta) \in r \textit{ with } \beta \in \mathsf{Attr}_{\bigcirc}(M,r)^i\} \\
&\qquad \cup \{wX\alpha \in \mathscr{S}^* \mid X \in \Box \textit{ and } \forall\ (wX\alpha, \beta) \in r \textit{ have } \beta \in \mathsf{Attr}_{\bigcirc}(M,r)^i\} \\
\mathsf{Attr}_{\bigcirc}(M,r) &= \bigcup_{i \in \mathbb{N}} \mathsf{Attr}_{\bigcirc}(M,r)^i.
\end{aligned}$$

Note that the attractor $\mathsf{Attr}_{\bigcirc} = \mathsf{Attr}_{\bigcirc}(\overline{\mathcal{L}(A)}, \Rightarrow)$ may contain sentential forms that do not appear in the game arena. The reason is that the attractor conducts a bottom-up search for the sentential forms from which refuter can enforce the derivation of a word $w \in \overline{\mathcal{L}(A)}$. Some words $w \in \overline{\mathcal{L}(A)}$ may already not be derivable from S in G or the attractor may include a sentential form α that is not derivable from S. For example if the grammar has rules $S_{\bigcirc} \to_G aX$, $X_{\bigcirc} \to_G b \mid c$ and $b, c \in \overline{\mathcal{L}(A)}$, then we have $X \in \mathsf{Attr}_{\bigcirc}$ although no node with label X is contained in the game arena.

But as we want to derive the winner of the context-free game $\mathscr{G}$, we are only interested in those labels. Consider thus the attractor $\mathsf{Attr}_{\bigcirc}(\overline{L}, \to_{\mathsf{B}_G})$, which is restricted to the sentential forms appearing in the game arena. The initial set $\overline{L}$ contains the nodes in the game arena B_G that are labeled with a terminal word, which are contained in $\overline{\mathcal{L}(A)}$, i.e. $\overline{L} = \{w \in T^* \mid S \Rightarrow^* w \text{ and } w \in \overline{\mathcal{L}(A)}\}$. Relation $\to_{\mathsf{B}_G}$ contains all edges from the game arena.

Refuter wins the context-free game $\mathscr{G}$ if and only if $S \in \mathsf{Attr}_{\bigcirc}(\overline{L}, \to_{\mathsf{B}_G})$ where S is the starting symbol. Fortunately, it is easy to show by induction that if a sentential form α appears in the game arena and is contained in $\mathsf{Attr}_{\bigcirc}$, then it is also part of $\mathsf{Attr}_{\bigcirc}(\overline{L}, \to_{\mathsf{B}_G})$ and vice versa. Therefore, we can decide the context-free game $\mathscr{G}$ from the saturated automaton $\mathscr{A}$. Refuter wins $\mathscr{G}$ if and only if $\mathscr{A}$ accepts the starting symbol S.

This approach was presented in [1] in order to solve pushdown game systems (PGS). A PGS is given by a pushdown system (PDS) $\mathscr{P}$ and an alternating $\mathscr{P}$-automaton.

The game is played on the configuration graph of the PDS. The player who owns the state is allowed to pick the transition rule to be applied. Refuter wins if the play on the configuration graph ends in a configuration that is not accepted by the alternating $\mathscr{P}$-automaton. An alternating $\mathscr{P}$-automaton can be understood as a finite automaton reading configurations. Otherwise prover wins.

Due to the similarities to context-free games, we can adapt this approach to our context. We take our grammar G as left-hand side of the inclusion instead of the PDS. For the right-hand side, we first determinize A into automaton A^{det}. Then we transform A^{det} into an alternating A^{det}-automaton which works similarly as the $\mathscr{P}$-automaton and initially accepts $\overline{\mathcal{L}(A)}$. The reader may wonder at this point whether we could directly saturate a finite automaton recognizing $\overline{\mathcal{L}(A)}$ or whether we could use a non-deterministic automaton to built the alternating automaton. The answer is negative, it is both necessary to have a deterministic automaton and to use the corresponding alternating automaton. We explain why this is important after the formal definition of the alternating A^{det}-automaton $\mathscr{A}$ and the introduction of the saturation rule.

An alternating A^{det}-automaton $\mathscr{A}$ has the same set of states Q^{det} as the finite automaton A^{det} and also reads terminal words. However the transitions are not of shape $q \xrightarrow{a} p$ for states $q, p \in Q^{\text{det}}$ and terminal a, but of shape $q \xrightarrow{a} F$, where F is a positive Boolean formula over the set of states Q^{det}. Thus, F is of shape $F = \bigvee_{i=1}^{n} K_i$, for clauses $K_i = \bigwedge_{j=1}^{m} p_{ij}$. In our setting we represent the disjunctions as non-determinism and conjunctions by macrostates (sets of states of A^{det}), i.e. instead of transition $q \xrightarrow{a} F$, we have transitions $q \xrightarrow{a} \bigcup_{j=1}^{m} p_{ij}$ for $i = 1, dots, n$. Intuitively, if we take a transitions $q \xrightarrow{a} \bigcup_{j=1}^{m} p_{ij}$, this means that we simultaneously move to all states $p_{i,1}, \ldots, p_{im}$. The usage of an alternating automaton allows us to represent the choices of the players in different ways. Provers choices are represented by macrostates and the ones of refuter by non-determinism.

Definition 15
Let A be the finite automaton given by the context-free game and $A^{det} = (T, Q^{det}, q_I^{det}, Q_F^{det}, \to_{det})$ be a deterministic automaton recognizing the same language. Then an alternating A^{det}-automaton $\mathscr{A}$ is given by the tuple $(T \cup N, \mathscr{Q}, \to_{\mathscr{A}}, q_I, \mathscr{Q}_F)$, where

- *$T \cup N$ is the input alphabet,*
- *$\mathscr{Q} = Q^{det}$ is the set of states,*
- *the set of transitions is of the form $\to_{\mathscr{A}} \subseteq \mathscr{Q} \times T \cup N \times 2^{\mathscr{Q}}$. We initialize $\to_{\mathscr{A}}$ by the following transitions: $p \to_{\mathscr{A}} \{q\}$ if $p \xrightarrow{a}_{det} q$.*
- *$q_I = q_I^{det}$ is the initial state.*
- *$\mathscr{Q}_F = \mathscr{Q} \setminus Q_F^{det}$ is the set of final states.*
- *Automaton $\mathscr{A}$ accepts a sentential form α if there exists a run $q_I \xrightarrow{\alpha}{}^{*}_{\mathscr{A}} P$ with $P \subseteq \mathscr{Q}_F$. Here $\to^*_{\mathscr{A}}$ represents the reflexive and transitive closure of $\to_{\mathscr{A}}$. It is defined by*

$$q \xrightarrow{\varepsilon}{}^{*}_{\mathscr{A}} \{q\} \text{ and}$$

$$q \xrightarrow{\gamma\alpha}{}^{*}_{\mathscr{A}} \bigcup_{i=1}^{n} P_i, \text{ if } q \xrightarrow{\gamma}_{\mathscr{A}} \{p_1, \ldots, p_n\} \text{ and } p_1 \xrightarrow{\alpha}{}^{*}_{\mathscr{A}} P_i \text{ for } i = 1, \ldots, n$$

The alternating automaton initially (before the saturation) accepts $\overline{\mathcal{L}(A)}$. Every word $w \in \overline{A}$ has an unique run $\mathcal{C}$ in A^{det} from $q_I = q_I^{\text{det}}$ to some state $q \notin Q_F^{\text{det}}$. Then w has an accepting run $q_I \xrightarrow{w}{}^{*}_{\mathscr{A}} \{q\} \subseteq \mathscr{Q}_F$ as we took over all the transitions $p \xrightarrow{a} q$ from $\mathcal{C}$ as transitions $p \xrightarrow{a}_{\mathscr{A}} \{q\}$ in $\mathscr{A}$.

The saturation rules by which we add transitions to $\mathscr{A}$ are as follows.

Definition 16
Let $p \in \mathscr{Q}$ and $X \to_G \alpha_1 \mid \cdots \mid \alpha_n$ be all rules in G with left-hand side X.

1. *If $X \in \bigcirc$ and $q \xrightarrow{\alpha_i}{}^{*}_{\mathscr{A}} P$ for some $i \in \{1, \dots, n\}$, we add transition $q \xrightarrow{X}_{\mathscr{A}} P$.*
2. *If $X \in \Box$ and $q \xrightarrow{\alpha_i}{}^{*}_{\mathscr{A}} P_i$ for some $i = 1, \dots, n$, we add transition $q \xrightarrow{X}_{\mathscr{A}} \bigcup_{i=1}^{n} P_i$.*

We apply these rules iteratively, starting with the alternating automaton $\mathscr{A} = \mathscr{A}_0$ containing only transitions for terminals and generate a sequence $\mathscr{A} = \mathscr{A}_0, \mathscr{A}_1, \mathscr{A}_2, \dots$ of alternating automata. In each iteration, we add transitions to $\mathscr{A}_i$ to generate $\mathscr{A}_{i+1}$ by considering each possible pair (q, X) for state $q \in Q^{\text{det}}$ and $X \in N$. Let $X \to_G \alpha_1 \mid \cdots \mid \alpha_n$ be all rules with left-hand side X. Then we add all transitions that we can generate from state q and rules $X \to_G \alpha_1 \mid \cdots \mid \alpha_n$ using the first saturation rule if $X \in \bigcirc$ and using the second saturation rule if $X \in \Box$. We call one iteration a **saturation step** and the whole iterations the **saturation process**. The saturation process ends after a finite number of steps as there exists only a finite number of possible transitions that an alternating automaton can have. We denote the final automaton in the sequence by $\mathscr{A}_{\text{attr}}$. As the name suggests, this automaton accepts $\mathsf{Attr}_{\bigcirc}$. We prove this at the end of the section.

The intuition behind the rules is the following.

We start the saturation process with automaton $\mathscr{A}_0$ which only contains transitions for the terminals. By construction, the state changes in $\mathscr{A}_0$ reflect the ones in A^{det}. The goal of the saturation is that final automaton $\mathscr{A}_{\text{attr}}$ accepts the sentential forms contained in $\mathsf{Attr}_{\bigcirc}$. Thus, it makes sense to have transition $q_0 \xrightarrow{X}_{\mathscr{A}} Q$ if refuter can enforce the derivation of some $w \in T^*$ s.t. $q_0 \xrightarrow{w} p \in Q$ in A^{det} in a play starting from X. We say that refuter can enforce state changes Q from state q_0 and non-terminal X. In particular, if $Q \subseteq Q_F$ then refuter can enforce the derivation of a word which is not contained $\mathcal{L}(A)$ and thus $X \in \mathsf{Attr}_{\bigcirc}$.

Suppose now for example that $X \in \bigcirc$ and $X \to_G \gamma_1\gamma_2$ is a grammar rule with $\gamma_1, \gamma_2 \in \mathscr{S}$. Then, refuter can enforce the derivation of the same terminal words in a play from X as in a play from $\gamma_1\gamma_2$. The words derivable from $\gamma_1\gamma_2$ are of a particular shape wv, where w is derivable from γ_1 and v from γ_2. Thus, if refuter can enforce state changes Q_1 from q_0 and γ_1, we need the state changes Q_p that she can enforce from each $p \in Q_1$ and γ_2. Finally, we get that refuter can enforce state changes $\bigcup_{p \in Q_1} Q_p$ from q_0 and $\gamma_1\gamma_2$.

The discussion implies that we can not only add transitions with left-hand side q_0. In general, we want to have a transition $q \xrightarrow{Y}_{\mathscr{A}} Q'$ if refuter can enforce state changes Q' from q and Y. Let us suppose for now that $\mathscr{A}$ already contains transitions $q_0 \xrightarrow{\gamma_1}_{\mathscr{A}} Q_1$ and $p \xrightarrow{\gamma_2}_{\mathscr{A}} Q_p$ for $p \in Q_1$. By the definition of runs $\longrightarrow^{*}_{\mathscr{A}}$ in the alternating automaton, we get $q_0 \xrightarrow{\gamma_1\gamma_2}{}^{*}_{\mathscr{A}} \bigcup_{p \in Q_1} Q_p$.

$$q_0 \xrightarrow{\gamma_1}_{\mathscr{A}} Q_1 \begin{cases} p_1 \xrightarrow{\gamma_2}_{\mathscr{A}} Q_{p_1} \\ \vdots \\ p_n \xrightarrow{\gamma_2}_{\mathscr{A}} Q_{p_n} \end{cases}$$

$$\wr$$

$$q_0 \xrightarrow{\gamma_1\gamma_2}{}^{*}_{\mathscr{A}} Q = \bigcup_{p \in Q_1} Q_p$$

Thus, we add transition $q_0 \xrightarrow{X}_{\mathscr{A}} Q$ to $\mathscr{A}$ if $q_0 \xrightarrow{\alpha_1\alpha_2}{}^{*}_{\mathscr{A}} Q$ as given by the first saturation rule.

In the case, where X belongs to prover, we need to take all rules $X \to \alpha$ ($\alpha \in \mathscr{S}^*$) with left-hand side X into account. Suppose that refuter can enforce state changes Q_α from q_0 and α i.e. we have $q_0 \xrightarrow{\alpha}{}^{*}_{\mathscr{A}} Q_\alpha$. Then refuter can only guarantee that the word w derived from

X induces state change $q_0 \xrightarrow{w} p \in \bigcup_{X\to\alpha} Q_\alpha$ as prover is allowed to pick the first rule. In this case, the second saturation rule dictates to add transition $q_0 \xrightarrow{X}_{\mathscr{A}} \bigcup_{X\to\alpha} Q_\alpha$ as desired.

In the final automaton $\mathscr{A}_{\text{attr}}$, the plays starting from non-terminal X are represented by all transitions with left-hand side q_0 which are labeled by X.

$$\begin{gathered} q_0 \xrightarrow{X}_{\mathscr{A}} Q_1 \\ \vdots \\ q_0 \xrightarrow{X}_{\mathscr{A}} Q_n. \end{gathered}$$

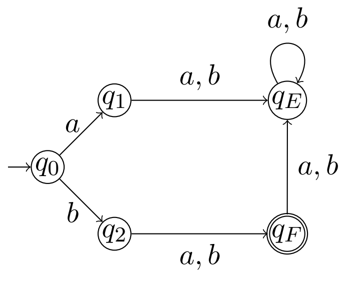
For all the words w derivable in a play from X holds that $q_0 \xrightarrow{w} q \in \bigcup_{i=1}^{n} Q_i$ in A^{det}. Concerning the alternating structure of the plays, refuter can guarantee that some terminal word w is derived with $q_0 \xrightarrow{w} p \in Q_i$ for any $i = 1, \ldots, n$. However, she can not narrow it down to a particular state $p \in Q_i$. This last choice belongs to prover. We encountered a similar flattening of the game to DNF-formulas σ_X in the summary approach.

4.1. Example

To promote a better understanding of the saturation approach, we demonstrate the saturation process on an example $\mathscr{G}_{ex}$. It is given by the context-free grammar $\mathcal{G}_{ex}$ and the finite automaton $\mathcal{A}_{ex}$ depicted in Figure 11.

$$\mathcal{G}_{ex}: \quad \begin{aligned} S_{\Box} &\to XY \\ X_{\Box} &\to a \mid b \\ Y_{\bigcirc} &\to a \mid b \end{aligned}$$

$\mathcal{A}_{ex}$:



Arena:

$$\begin{tikzcd} & & S_{\Box} \arrow[d] & & \\ & & X_{\Box}Y \arrow[dl] \arrow[dr] & & \\ & aY_{\bigcirc} \arrow[dl] \arrow[d] & & bY_{\bigcirc} \arrow[d] \arrow[dr] & \\ aa & ab & & ba & bb \end{tikzcd}$$

Figure 11.: Example $\mathscr{G}_{ex}$.

The nodes with labels in $M_1 = \{aa, ab, aY\}$ in the game arena are contained in the attractor $\mathsf{Attr}_{\bigcirc}$, while the labels in $M_2 = \{ba, bb, bY, XY, S\}$ are not. We will see that after the saturation process, the sentential forms in M_1 will be recognized by $\mathscr{A}_{\text{attr}}$, whereas the ones in M_2 are not accepted. The following table depicts which transitions are added in each iteration and from which of the two saturation rules they stem. For some of the added transitions, we explain in more detail how they were created. The initial transitions of the alternating automaton $\mathscr{A}_0$ are given by

$$\begin{aligned} q_0 &\xrightarrow{a}_{\mathscr{A}} \{q_1\} & q_0 &\xrightarrow{b}_{\mathscr{A}} \{q_2\} \\ q_1 &\xrightarrow{x}_{\mathscr{A}} \{q_E\} & q_1 &\xrightarrow{x}_{\mathscr{A}} \{q_E\} \\ q_F &\xrightarrow{x}_{\mathscr{A}} \{q_E\} & q_E &\xrightarrow{x}_{\mathscr{A}} \{q_E\} \end{aligned}$$

where $x \in \{a, b\}$.

	rule 1	rule 2
$\mathscr{A}_1$	$q_0 \xrightarrow{Y}_{\mathscr{A}} \{q_1\}$ $q_0 \xrightarrow{Y}_{\mathscr{A}} \{q_2\}$	$q_0 \xrightarrow{X}_{\mathscr{A}} \{q_1, q_2\}$
	$q_1 \xrightarrow{Y}_{\mathscr{A}} \{q_E\}$	$q_1 \xrightarrow{X}_{\mathscr{A}} \{q_E\}$
	$q_2 \xrightarrow{Y}_{\mathscr{A}} \{q_F\}$	$q_2 \xrightarrow{X}_{\mathscr{A}} \{q_F\}$
	$q_F \xrightarrow{Y}_{\mathscr{A}} \{q_E\}$	$q_F \xrightarrow{X}_{\mathscr{A}} \{q_E\}$
	$q_E \xrightarrow{Y}_{\mathscr{A}} \{q_E\}$	$q_E \xrightarrow{X}_{\mathscr{A}} \{q_E\}$
$\mathscr{A}_2$		$q_0 \xrightarrow{S}_{\mathscr{A}} \{q_F, q_E\}$
		$q_1 \xrightarrow{S}_{\mathscr{A}} \{q_E\}$
		$q_2 \xrightarrow{S}_{\mathscr{A}} \{q_E\}$
		$q_F \xrightarrow{S}_{\mathscr{A}} \{q_E\}$
		$q_E \xrightarrow{S}_{\mathscr{A}} \{q_E\}$

Let us examine some of the new transitions more closely and explain exactly how they were added to $\mathscr{A}$.

- $q_1 \xrightarrow{Y}_{\mathscr{A}} \{q_E\}$ in $\mathscr{A}_1$
  This transition was added due by the first saturation rule due to the path $q_1 \xrightarrow{a}{}^{*}_{\mathscr{A}} \{q_E\}$ (or equally $q_1 \xrightarrow{b}{}^{*}_{\mathscr{A}} \{q_E\}$)in $\mathscr{A}_0$.

- $q_0 \xrightarrow{X}_{\mathscr{A}} \{q_1, q_2\}$ in $\mathscr{A}_1$
  Here, we used the first saturation rule and the paths $q_1 \xrightarrow{a}{}^{*}_{\mathscr{A}} \{q_1\}$, $q_1 \xrightarrow{b}{}^{*}_{\mathscr{A}} \{q_2\}$ in $\mathscr{A}_0$. By the newly added transition, sentential form X is accepted. Note that X is contained in $\mathsf{Attr}_{\bigcirc}$, but not in $\mathsf{Attr}_{\bigcirc}()$.

- $q_0 \xrightarrow{S}_{\mathscr{A}} \{q_E, q_F\}$ in $\mathscr{A}_1$
  This time, we used the second saturation rule with path $q_0 \xrightarrow{X}{}^{*}_{\mathscr{A}} \{q_E, q_F\}$, which consists of the transitions $q_0 \xrightarrow{X}_{\mathscr{A}} \{q_1, q_2\}$, $q_1 \xrightarrow{Y}_{\mathscr{A}} \{q_E\}$ and $q_2 \xrightarrow{Y}_{\mathscr{A}} \{q_F\}$. Note that the new transition is also a rejecting path for S. Thus S is not accepted by $\mathscr{A}_{\text{attr}} = \mathscr{A}_2$ as desired.

4.2. Finite automaton and determinism

We address the two questions whose answers were postponed above.

The first question was whether we can saturate a finite automaton recognizing $\overline{\mathcal{L}(A)}$ instead of an alternating one. The issue when using a finite automaton is that we cannot encode the choices of prover and refuter differently in the saturated automaton. By saturating an alternating automaton, we can represent refuters choices in the automaton by non-determinism and provers by transitions to sets of states. If we have rules $X \to \alpha_1 \mid \cdots \mid \alpha_n$ and paths $q \xrightarrow{\alpha_i}{}^{*}_{\mathscr{A}} P_i$, then we add transitions $q \xrightarrow{X}{}^{*}_{\mathscr{A}} P_i$ for each $i = 1, \ldots, n$ if X belongs to refuter. If X belongs to prover however, we only add transition $q \xrightarrow{X}{}^{*}_{\mathscr{A}} \bigcup_{i=1}^{n} P_i$.

In a finite automaton, we cannot use sets of states to represent the choices of prover and would have to use the same saturation rule for both cases. We can easily construct an example (Figure 12), where the saturated automaton does not accept the attractor $Attr_{\bigcirc}(\overline{\mathcal{L}(A)})$. The grammar of the context-free game only consists of rule $X_{\square} \to a \mid b$ and the automaton $\overline{A}$ accepting $\overline{\mathcal{L}(A)}$ is depicted on the upper right. The saturation is finished after only one step and the resulting automaton is $\overline{A}_1$.

$G: \quad X_{\square} \to a \mid b$

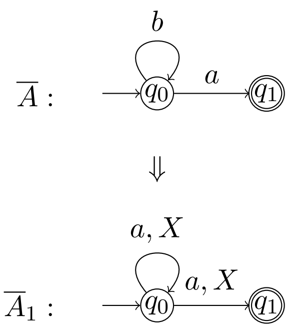
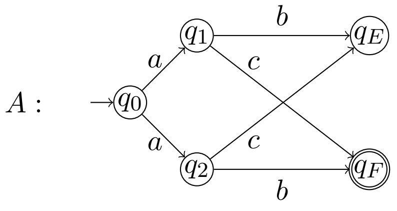


Figure 12.: Saturation does not work with finite automata.

The automaton $\overline{A}_1$ accepts the starting symbol X, although it is clearly not contained in the attractor. Thus the answer to the first question is negative.

The second question is whether we can create the alternating automaton from a non-deterministic finite automaton. This would be very beneficial for the runtime complexity as determinizing an automaton entails a guaranteed exponential blow-up of the complexity. Unfortunately, this step cannot be avoided as the following counter-example shows (Figure 13).

$$G: \quad \begin{array}{l} S_{\bigcirc} \to_G aX \\ X_{\bigcirc} \to_G b \mid c \end{array}$$



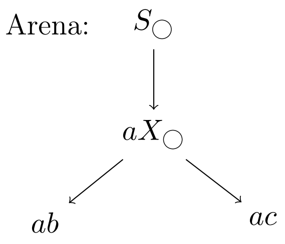


Figure 13.: Saturation does not work with non-deterministic automata.

All the derivable words are accepted by the automaton A. Thus, S is not contained in the attractor $\mathsf{Attr}_{\bigcirc}$. We construct the alternating A-automaton as in Definition 15. We handle the non-determinism as provers choice i.e. if $q \xrightarrow{a} p_i$ for $i = 1, \ldots, n$ are all the edges starting from state q with label a in A, then we have transition $q \xrightarrow{a}_{\mathscr{A}} \{p_1, \ldots, p_n\}$ in $\mathscr{A}$. Intuitively, it makes sense to let prover resolve the non-determinism because if there exists an accepting path for a word in the automaton A, then the word should be accepted. As prover aims for acceptance of the derived word, she always picks the accepting path. Refuter on the other hand would rather select the non-accepting paths and thereby falsify the outcome of the game. But the following saturation process shows that the resulting automaton does not always recognize the attractor $\mathsf{Attr}_{\bigcirc}$.

	added rules
$\mathscr{A}_0$	$q_0 \xrightarrow{a}_{\mathscr{A}} \{q_1, q_2\}$ (non-determinism)
	$q_1 \xrightarrow{b}_{\mathscr{A}} \{q_E\}$ $q_1 \xrightarrow{c}_{\mathscr{A}} \{q_F\}$
	$q_2 \xrightarrow{b}_{\mathscr{A}} \{q_F\}$ $q_2 \xrightarrow{c}_{\mathscr{A}} \{q_E\}$
$\mathscr{A}_1$	$q_1 \xrightarrow{X}_{\mathscr{A}} \{q_F\}$, $q_1 \xrightarrow{X}_{\mathscr{A}} q_E\}$
	$q_2 \xrightarrow{S}_{\mathscr{A}} \{q_F\}$, $q_2 \xrightarrow{X}_{\mathscr{A}} q_E\}$
$\mathscr{A}_2$	$q_0 \xrightarrow{S}_{\mathscr{A}} \{q_F, q_E\}$
	$q_0 \xrightarrow{S}_{\mathscr{A}} \{q_F\}$, $q_0 \xrightarrow{S}_{\mathscr{A}} q_E\}$

In A_2, we created transitions $q_0 \xrightarrow{S}_{\mathscr{A}} \{q_E, q_F\}$, $q_0 \xrightarrow{S}{}^{*}_{\mathscr{A}} \{q_F\}$ and $q_0 \xrightarrow{S}{}^{*}_{\mathscr{A}} \{q_E\}$ in $\mathscr{A}$ by the first saturation rule. The last transition $q_0 \xrightarrow{S}_{\mathscr{A}} \{q_E\}$ is an accepting run for S given that the final states of $\mathscr{A}$ are exactly the non-accepting ones in A. Thus, the resulting automaton suggests that S is contained in $\mathsf{Attr}_{\bigcirc}$.

The problem is that by letting prover resolve the non-determinism, we give refuter the opportunity to react to provers choice. If prover resolves the non-determinism to q_1, refuter chooses transition $q_1 \xrightarrow{X}_{\mathscr{A}} q_E$ corresponding to rule $X \to_G b$ and if prover chooses q_2, refuter reacts by picking transition $q_2 \xrightarrow{X}_{\mathscr{A}} q_E$ corresponding to $X \to_G c$.

This issue is related to the difference between bisimulation equivalence and language inclusion. By asking prover to resolve the non-determinism of A, she actually has to prove a stronger claim than language inclusion $\mathcal{L}(G) \subseteq \mathcal{L}(A)$. Prover will only win if the automaton A can simulate each branching behavior of the game arena by an accepting path. If there exists such a simulation, language inclusion also holds. But the converse does not hold. Even if $\mathcal{L}(G) \subseteq \mathcal{L}(A)$, the automaton A may not be able to simulate the behavior of the game arena by an accepting path. This is the case in our example.

In the first step, the game arena moves from sentential form S to aX. Prover can either simulate this behavior in A by moving to q_1 or q_2. In case prover picked q_1 (q_2), the game arena can move to ab (ac). Then, prover can only simulate this behavior by moving to q_E which is non-accepting state. Thus, not all behaviors of the game arena can be simulated by an accepting run in A, although language inclusion holds.

We avoid this situation by determinizing automaton A.

4.3. Correctness of the saturation approach

This section is dedicated to prove the correctness of the presented approach. More precisely, we prove the following theorem.

Theorem 1
Let $\mathscr{G}$ be a context-free game given by context-free grammar G and finite automaton A. Let furthermore be A^{det} be a deterministic automaton recognizing $\overline{\mathcal{L}(A)}$ and $\mathscr{A}$ the corresponding alternating A^{det}-automaton. Then, $\mathscr{A}_{attr}$ accepts exactly the sentential forms in $\mathsf{Attr}_{\bigcirc}$.

There is a similar theorem (Theorem 1.3.1) in [2] for pushdown game systems. The proofs of both theorems is nearly identical. But for the sake of completeness, we state the adapted proof here.

Proof. Let $\mathscr{A}_1, \mathscr{A}_2, \ldots, \mathscr{A}_m = \mathscr{A}_{\text{attr}}$ be the intermediary alternating automata created during the saturation process. We assume that we added only one transition in each step, i.e. $\mathscr{A}_{i+1} = \mathscr{A}_i \cup \{p \xrightarrow{A}_{\mathscr{A}} P\}$ for some non-terminal A, state $q \in \mathscr{Q}$ and set of states $P \in 2^{\mathscr{Q}}$. Note that this assumption does not match with the definition of our saturation process. We add all possible transitions for each pair (q, X) of a state $q \in \mathscr{Q}$ and a non-terminal X in one saturation step. But the alternating automaton $\mathscr{A}_{\text{attr}}$ reached at the fixed-point of the saturation process is the same for both definitions. Adding only one transition in each iteration simplifies the proof, therefore we use the alternative definition of the saturation process.

For this proof, we need to examine intermediary points of runs. From the last bullet in Definition 15 it is clear that the intermediary points have to be set of states. Thus, we generalize $\longrightarrow_{\mathscr{A}}$ to sets of states P on the left-hand side.

$$P \xrightarrow{\varepsilon}_{\mathscr{A}} P$$
$$P \xrightarrow{\gamma}_{\mathscr{A}} \bigcup_{q\in P} P_q, \text{ if } q \xrightarrow{\gamma}_{\mathscr{A}} P_q \text{ for each } q \in P \text{ where } \gamma \in \mathscr{S}.$$

Then we can write each run $q \xrightarrow{\alpha}{}^*_{\mathscr{A}} S$ as

$$q \xrightarrow{\alpha_1}_{\mathscr{A}} S_1 \xrightarrow{\alpha_2}_{\mathscr{A}} S_2 \xrightarrow{\alpha_2}_{\mathscr{A}} \dots \xrightarrow{\alpha_n}_{\mathscr{A}} S_n = S$$

for some $S_i \in 2^{\mathscr{Q}}$ $(i = 1, \dots, n)$. Similarly, we can generalize $\longrightarrow^*_{\mathscr{A}}$ to sets of states on the left-hand side.

$$P \xrightarrow{\varepsilon}{}^*_{\mathscr{A}} P$$
$$P \xrightarrow{\alpha}{}^*_{\mathscr{A}} \bigcup_{q\in P} P_q, \text{ if } q \xrightarrow{\alpha}{}^*_{\mathscr{A}} P_q \text{ for each } q \in P \text{ where } \alpha \in \mathscr{S}^*.$$

We now show the claim in two steps by Lemma 1 and Lemma 3.

- $\mathsf{Attr}^i_{\bigcirc} \subseteq \mathcal{L}(\mathscr{A}_{\text{attr}})$ for $i \in \mathbb{N}$
- $\mathcal{L}(\mathscr{A}_i) \subseteq \mathsf{Attr}_{\bigcirc}$ for $i = 0, \dots, m$

The, the claim of the theorem follows as $\mathsf{Attr}_{\bigcirc} = \bigcup_{i\geq 0} \mathsf{Attr}^i_{\bigcirc}$ and $\mathcal{L}(\mathscr{A}_{\text{attr}}) = \bigcup_{i\geq 0} \mathcal{L}(\mathscr{A}_i)$. □

Step 1:

Lemma 1
$\mathsf{Attr}^i_{\bigcirc} \subseteq \mathcal{L}(\mathscr{A}_{attr})$ *for* $i \in \mathbb{N}$

Proof. We prove the claim by induction over i.

Base case $i = 0$**:**

We initialize the alternating automaton $\mathscr{A}$ s.t. it recognizes exactly $\overline{\mathcal{L}(A)}$. Then, we have that

$$\mathsf{Attr}^0_{\bigcirc} = \overline{\mathcal{L}(A)} = \mathscr{A} \subseteq \mathcal{L}(\mathscr{A}_{\text{attr}}).$$

By adding transitions, we only augment the language recognized by the alternating automaton. Therefore, the last inclusion holds.

Induction step $i \to i + 1$**:**

Assume it already holds that $\mathsf{Attr}^i_{\bigcirc} \subseteq \mathcal{L}(\mathscr{A}_{\text{attr}})$. We consider some sentential form $wX\alpha \in \mathsf{Attr}^{i+1}_{\bigcirc} \setminus \mathsf{Attr}^i_{\bigcirc}$ and show that $wX\alpha \in \mathcal{L}(\mathscr{A}_{\text{attr}})$.

We need to distinguish two cases.

- $X \in \bigcirc$

Then, there must exist a rule $X \to \beta$ in G s.t. $wX\alpha \Rightarrow_G w\beta\alpha$ with $w\beta\alpha \in \mathsf{Attr}^i_{\bigcirc}$. By the induction hypothesis, this means that $\mathscr{A}_{\text{attr}}$ accepts $w\beta\alpha$. Therefore, there must be an accepting path for $w\beta\alpha$ in $\mathscr{A}_{\text{attr}}$:

$$q_I \xrightarrow{w}{}^{*}_{\mathscr{A}} S_1 \xrightarrow{\beta}{}^{*}_{\mathscr{A}} S_2 \xrightarrow{\alpha}{}^{*}_{\mathscr{A}} S \subseteq \mathscr{Q}_F$$

From $S_1 \xrightarrow{\beta}{}^{*}_{\mathscr{A}} S_2$ we get by definition of $\longrightarrow^{*}_{\mathscr{A}}$ that there exist a set of states S_q for each $q \in S_1$ s.t. $q \xrightarrow{\beta}{}^{*}_{\mathscr{A}} S_q$ and $S_2 = \bigcup_{q \in S_1} S_q$. But then, we know by the first saturation rule that $\mathscr{A}_{\text{attr}}$ contains transitions $q \xrightarrow{X}_{\mathscr{A}} S_q$ for each $q \in S_1$. Using these transitions, we can construct the following accepting path in $\mathscr{A}_{\text{attr}}$.

$$q_I \xrightarrow{w}{}^{*}_{\mathscr{A}} S_1 \xrightarrow{X}_{\mathscr{A}} S_2 \xrightarrow{\alpha}{}^{*}_{\mathscr{A}} S \subseteq \mathscr{Q}_F$$

Thus, $wX\alpha \in \mathcal{L}(\mathscr{A}_{\text{attr}})$.

- $X \in \square$ Let $X \to \beta_1 \mid \cdots \mid \beta_n$ be all the rules with left-hand side X. If $wX\alpha \in \mathsf{Attr}^{i+1}_{\bigcirc}$, then $w\beta_1\alpha$ for all $i \in \{1, \dots, n\}$ must be contained in $\mathsf{Attr}^{i+1}_{\bigcirc}$. Then, we have accepting paths

$$q_I \xrightarrow{w}{}^{*}_{\mathscr{A}} S_1 \xrightarrow{\beta_i}{}^{*}_{\mathscr{A}} S_2^{(i)} \xrightarrow{\alpha}{}^{*}_{\mathscr{A}} S^{(i)} \subseteq \mathscr{Q}_F.$$

As in the first case, we can derive from $S_1 \xrightarrow{\beta_i}{}^{*}_{\mathscr{A}} S_2^{(i)}$ that there exist set of states $S_q^{(i)} \in 2^{\mathscr{Q}}$ s.t. $q \xrightarrow{\beta_i}{}^{*}_{\mathscr{A}} S_q^{(i)}$ and $S_2^{(i)} = \bigcup_{q \in S_1} S_q^{(i)}$ fir $i = 1, \dots, n$. Applying the second saturation rule adds transitions $q \xrightarrow{X}{}^{*}_{\mathscr{A}} \bigcup_{i=1}^{n} S_q^{(i)}$ to $\mathscr{A}$. This creates accepting path

$$q_I \xrightarrow{w}{}^{*}_{\mathscr{A}} S_1 \xrightarrow{X}{}^{*}_{\mathscr{A}} \bigcup_{q \in S_1} \bigcup_{i=1}^{n} S_q^{(i)} \xrightarrow{\alpha}{}^{*}_{\mathscr{A}} \bigcup_{i=1}^{n} S^{(i)} \subseteq \mathscr{Q}_F.$$

for $wX\alpha$ and thus $wX\alpha \in \mathcal{L}(\mathscr{A}_{\text{attr}})$ as desired.

$\square$

Step 2:

Before we prove that $\mathcal{L}(\mathscr{A}_i) \subseteq \mathsf{Attr}_{\bigcirc}$ for $i = 0, \dots, m$, we show the following helping lemma.

Lemma 2
If refuter can enforce the derivation of some set $M = \{\beta_1, \dots, \beta_n\}$ of sentential forms from sentential form α, then $\alpha \in \mathsf{Attr}_{\bigcirc}(M, \Rightarrow)$.

Proof. If refuter can enforce the derivation of M from α, then she has a strategy $\mathsf{s}_{\bigcirc}$ s.t. any play conform to $\mathsf{s}_{\bigcirc}$ starting in α ends in M.

Towards a contradiction, we assume that $\alpha \notin \mathsf{Attr}_{\bigcirc}(M, \Rightarrow)$. We inductively construct a play π starting in α conform to $\mathsf{s}_{\bigcirc}$ s.t. no position of π is contained in the attractor. This in particular means that the play can not end in a sentential form contained in M.

Base case: We have $\alpha \notin \mathsf{Attr}_{\bigcirc}(M, \Rightarrow)$ by assumption.

Induction step: Let α_i be the last position of the play π_i that we constructed so far. By the induction hypothesis, $\alpha_i \notin \mathsf{Attr}_{\bigcirc}(M, \Rightarrow)$. We show that we can prolong π_i by a sentential form α_{i+1} s.t. the move is conform to $\mathsf{s}_{\bigcirc}$ and $\alpha_{i+1} \notin \mathsf{Attr}_{\bigcirc}(M, \Rightarrow)$. We need to distinguish two cases.

- $\alpha_i \in \bigcirc$
  As $\alpha_i \notin \mathsf{Attr}_{\bigcirc}(M, \Rightarrow)$, none of its successors wrt. $\Rightarrow$ is contained in the attractor. We choose $\alpha_{i+1} = \mathsf{s}_{\bigcirc}(\alpha_1, \ldots, \alpha_i)$.

- $\alpha_i \in \Box$
  As $\alpha_i \notin \mathsf{Attr}_{\bigcirc}(M, \Rightarrow)$, at least one of its successors wrt. $\Rightarrow$ is not contained in the attractor. We choose an arbitrary one of the successors that is not contained in $\mathsf{Attr}_{\bigcirc}(M, \Rightarrow)$ as α_{i+1}.

$\Box$

With the helping lemma at hand, we can prove Lemma 3

Lemma 3
$\mathcal{L}(\mathscr{A}_i) \subseteq \mathsf{Attr}_{\bigcirc}$ *for* $i = 0, \ldots, m$

Proof. We actually prove a stronger claim here. We show that if there is a run $q \xrightarrow{\alpha}{}^{*}_{\mathscr{A}} S$ for any $q \in \mathscr{Q}$, $S \subseteq \mathscr{Q}$, then refuter can enforce the derivation of a terminal word w starting from α which is contained in $M = \{w \in T^* \mid \exists p \in S \text{ s.t. } q \xrightarrow{w} p \text{ in } \overline{A^{\text{det}}\}}$. If we take $\alpha \in \mathcal{L}(\mathscr{A}_i)$, we get that $q = q_I$ and $S \subseteq \mathscr{Q}_F$. Then, it follows that $w \in M \subseteq \overline{\mathcal{L}(A)}$. By the helping Lemma 2, we also have that $\alpha \in \mathsf{Attr}_{\bigcirc}$.

As $M \subseteq \overline{\mathcal{L}(A)}$, we also have that $\alpha \in \mathsf{Attr}_{\bigcirc}(M, \Rightarrow) \subseteq \mathsf{Attr}_{\bigcirc}$.

We prove the claim above by induction over i.

Base case $i = 0$**:** Suppose we have a path $q \xrightarrow{\alpha}{}^{*}_{\mathscr{A}} S$ for some $q \in \mathscr{Q}$ and $\alpha \in \mathscr{S}^*$. As all the transitions in $\mathscr{A}_0$ correspond to edges in A^{det}, α must be a terminal word and S consists of only one state, say $S = \{p\}$. The claim thus follows trivially.

Induction step $i \to i+1$**:** As declared above, $\mathscr{A}_{i+1}$ was created from $\mathscr{A}_i$ by adding one transition. Let the newly added transition be $t = p_0 \xrightarrow[i+1]{X}_{\mathscr{A}} S_0$. By $\xrightarrow[i+1]{}_{\mathscr{A}}$, we denote the transitions of $\mathscr{A}_{i+1}$. Suppose now we have a run $q \xrightarrow[i+1]{\alpha}_{\mathscr{A}} S$.

We conduct an induction over the number of times j the new transition is used in the run.

If $j = 0$, the new transition t is never used in the run $q \xrightarrow[i+1]{\alpha}_{\mathscr{A}} S$ and we get the claim by the induction hypothesis on i.

Suppose now that t is used $j+1$ times.

Then, we can decompose the run $q \xrightarrow[i+1]{\alpha}{}^{*}_{\mathscr{A}} S$ into the following parts. Let $\alpha = \alpha_1 X \alpha_2$.

$$\begin{tikzcd}
q \arrow[r, "\alpha_1", "i"'] & S_1 \arrow[r, "X", "i+1"'] \arrow[d, phantom, "\ni"] & S_2 \arrow[r, "\alpha_2", "i+1"'] \arrow[d, phantom, "\supseteq"] & S \subseteq \mathscr{Q}_F \\
& p_0 \arrow[r, "X", "i+1"'] & S_0 &
\end{tikzcd} \tag{0.1}$$

In this run, the new transition T is used for the first time in $S_1 \xrightarrow[i+1]{X}_{\mathscr{A}} S_2$. We have that $p_0 \in S_1$ and $S_0 \subseteq S_2$.

We examine the parts of this run separately, apply the induction hypothesis to each one and combine the information at the end of the proof.

1. $q \xrightarrow[i]{\alpha_1}{}^{*}_{\mathscr{A}} S_1$
   This run does not use the new transition t. Thus, we can apply the induction hypothesis and get that refuter can enforce the derivation of some terminal w_1, starting from α_1, where $w_1 \in \{w \in T^* \mid \exists p \in S_1 \text{ s.t. } q \xrightarrow{w} p \text{ in } A^{\text{det}}\}$

2. $S_1 \setminus \{p_0\} \xrightarrow[i]{X}_{\mathscr{A}} S_2' \xrightarrow[i+1]{\alpha_2}{}^{*}_{\mathscr{A}} S'$.
   We consider the second part of the run in (0.1) where we cut out the new transition t from $S_1 \xrightarrow[i+1]{X}_{\mathscr{A}} S_2$. Thus, we have that $S_2' \subseteq S_2$ and $S' \subseteq S$ and the new transition t is used at most j times.

   For each of the states p in $S_1 \setminus \{p_0\}$, we have a run $p \xrightarrow[i]{X}_{\mathscr{A}} S_2'' \xrightarrow[i+1]{\alpha_2}{}^{*}_{\mathscr{A}} S''$ for some $S_2'' \subseteq S_2' \subseteq S_2$ and $S'' \subseteq S' \subseteq S$. In each of these runs, the new transition is used at most j times as well. Using the induction hypothesis, we get that refuter can enforce the derivation of some terminal w_2 from $X\alpha_2$ s.t. $w_2 \in \{w \in T^* \mid \exists p \in S'' \subseteq S \text{ s.t. } q \xrightarrow{w} p \text{ in } A^{\text{det}}\}$

3. $p_0 \xrightarrow[i+1]{X}_{\mathscr{A}} S_0$.
   It remains to examine the new transition t. From the run in (0.1), we get that $S_0 \subseteq S_2$. Transition t been included in $\xrightarrow[i+1]{}_{\mathscr{A}}$ by either the first or the second saturation rule. We distinguish the two cases.

   a) $X \in \bigcirc$
      The new transition was added by the first saturation rule from some grammar rule $X \to \beta$ and transition $p_0 \xrightarrow[i]{\beta}{}^{*}_{\mathscr{A}} P_0$. Let us consider the following run.

$$p_0 \xrightarrow[i]{\beta}{}^{*}_{\mathscr{A}} P_0 \xrightarrow[i+1]{\alpha_2}{}^{*}_{\mathscr{A}} U_0$$

      As $P_0 \subseteq S_2$, we also have that $U_0 \subseteq S$. In this run, we use t at most j times. Applying the induction hypothesis gives us that refuter can enforce the derivation of some x_0 from $\beta\alpha_2$ s.t. $x_0 \in \{w \in T^* \mid \exists p \in U_0 \subseteq S \text{ s.t. } q \xrightarrow{w} p \text{ in } A^{\text{det}}\}$.

   b) $X \in \square$
      The new transition was included by the second saturation rule from the grammar rules $X \to \beta_1 \mid \cdots \mid \beta_n$ with right-hand side X and corresponding transitions $p_0 \xrightarrow[i]{\beta_i}{}^{*}_{\mathscr{A}} S_0^{(i)}$, where $S_0 = \bigcup_{i=1}^{n} S_0^{(i)}$. As in the previous case, we get for the runs

$$p_0 \xrightarrow[i]{\beta_i}{}^{*}_{\mathscr{A}} P_0 \xrightarrow[i+1]{\alpha_2}{}^{*}_{\mathscr{A}} U_i$$

      that $U_i \subseteq S$ for $i = 1, \ldots, n$ and that they use the new transition t at most j times. We apply the induction hypothesis to get that refuter can enforce the derivation of some terminal word x_i from $\beta_i\alpha_2$ s.t. $x_i \in \{w \in T^* \mid \exists p \in U_i \subseteq S \text{ s.t. } q \xrightarrow{w} p \text{ in } A^{\text{det}}\}$.

Now, we combine the previous statements to prove the claim.
From 1., we get that refuter can enforce the derivation of some w_1 from α_1 s.t. $q \xrightarrow{w_1} p_1 \in S_1$ in A^{det}. There are two possible cases.

- $p_1 \neq p_0$
  In this case, we can use 2.: Refuter can enforce the derivation of some w_2 from $X\alpha_2$ s.t. $p_1 \xrightarrow{w_2} p_2 \in S$ in A^{det}. Recombining the facts gives us that prover can enforce the derivation of some w_1w_2 from $\alpha_1X\alpha_2$ by first deriving $w_1X\alpha_2$ from $\alpha_1X\alpha_2$ and then w_1w_2 from $w_1X\alpha_2$. Here, we use the following trivial observation.
  Observation 1. If player $\star \in \{\bigcirc, \square\}$ can enforce the derivation of some w from sentential form α, then she can also enforce the derivation of

  – vw from $v\alpha$ and

  - $w\beta$ from $\alpha\beta$.

  This proves the claim as we have that $q \xrightarrow{w_1} p_1 \xrightarrow{w_2} p_2 \in S$.

- $p_1 = p_0$
  We use 3.a) if X belongs to refuter resp. 3.b) if X belongs to prover.
  - $X \in \bigcirc$
    Using 3.a), we get that refuter can enforce the derivation of some x_0 from β where β is the right-hand side of some grammar rule $X \to \beta$ s.t. $p_0 \xrightarrow{x_0} p_3 \in S$ in A^{det}. But the, prover can also enforce some $w_1 x_0$ from $\alpha_1 X \alpha_2$ by first deriving $w_1 X \alpha_1$, then applying rule $X \to \beta$ and finally deriving $w_1 x_0$ from $w_1 \beta \alpha_2$. Then, the claim follows from $q \xrightarrow{w_1} p_0 \xrightarrow{x_0} p_3 \in S$.
  - $X \in \Box$
    By 1., we got that refuter can enforce some $w_1 X \alpha_1$ from $\alpha_1 X \alpha_2$. As $X \in \Box$, she is allowed to derive any $w_1 \beta_i \alpha_2$ from $w_1 X \alpha_2$. From 3.b) we get that for any of these $w_1 \beta_i \alpha_2$, refuter can enforce the derivation of some x_i s.t. $p_0 \xrightarrow{x_i} p_4 \in S$ in A^{det}. Thus, refuter can enforce some $w_1 x_i$ from $\alpha_1 X \alpha_2$ and it holds that $q \xrightarrow{w_1} p_0 \xrightarrow{x_i} p_3 \in S$. Thus, the claim follows.

$\Box$

4.4. Overview of the saturation method

To end the section about the saturation approach, we present a brief overview of the algorithm to solve context-free games. Let $\mathscr{G}$ be a context-free game given by the context-free grammar G and the finite automaton A. We employ the following steps.

1. Determinize the finite automaton A into A^{det}.
2. Construct the alternating A^{det}-automaton $\mathscr{A}$ as in Definition 15.
3. Saturate the automaton $\mathscr{A}$ into $\mathscr{A}_{\text{attr}}$ by the two saturation rules until no more transitions can be added.
4. Check whether the starting symbol S of the grammar is accepted by $\mathscr{A}_{\text{attr}}$. If this is the case, refuter wins $\mathscr{G}$, otherwise prover wins.

5. Guess & Check Approach

The idea of the *Guess & Check approach* is to reduce the inclusion game to a reachability game on a finite graph, which we call *GC game*. This finite graph and the game are closely related to the game arena with the corresponding reachability game, but we restrict the information (sentential forms) stored in each node to make the graph finite. The information is sufficient to simulate a play from $\mathcal{G}$ in the *GC game* and to preserve the winner of the inclusion game i.e. :

- Refuter has a winning strategy in the original game if and only if refuter has a winning strategy in the finite game.
- A winning strategy in the original game can be converted to a winning strategy for the *GC game* and vice versa.

This approach was proposed in [10] for pushdown game systems. Due to the similarities of pushdown game systems (PGS) and context-free games, we can use the method in our context after a few modifications.

The rest of this section is dedicated to the formal definition of the *GC game* and the reduction i.e. the conversion of winning strategies from one game to the other. But before diving into the definitions and proofs, we provide some intuition about the *GC game*.

5.1. The idea behind the *GC game*

The challenge is to reduce the (possibly) infinite game arena to the finite graph on which the *GC game* is played. This means that it is not possible to store the whole sentential form in the nodes. We have to adress the following two problems:

On one hand, we may not have a bound on the length of the terminal prefix of the sentential forms. Fortunately, this problem can be solved by simply storing the state change that the terminal prefix induces on A from q_0 on instead of the prefix itself. We emphasize the importance of the singular in the previous sentence. As in the saturation approach, we need to have a deterministic automaton on the right-hand side of the inclusion for the *Guess & Check approach*. The reason is actually the same as for the saturation approach, but we will get into details after the formal definition of the *GC game*.

On the other hand, the fact that we also may not have a bound on the length of the suffix of the sentential form is a problem. For example, this occurs if the grammar contains a rule of shape $X \rightarrow_G XX$. Thus, we can also not store the whole suffix in the node. This problem is more involved, but we can make use of the following consideration.

Suppose that we are at position $wX\alpha$. Then, the portion of the play starting from $wX\alpha$ to the first position of shape $ww'\alpha$, called subplay in the following, does not depend on α. The suffix α becomes important only from position $ww'\alpha$. Actually, the only information needed from the portion $wX\alpha$ to $ww'\alpha$ of the play is the terminal word v' that has been derived.

We use this insight to change the evolution of the plays. At a sentential form $wX\alpha$, refuter proposes a set of terminal words that may be derived from X in form of the state changes that they induce on A from q_w on. State q_w is the unique target state of the run from A on w from q_0 on. As the other player also influences the derivation process of X, prover usually can not narrow down the derived terminal word to a particular one. Thus he needs to propose a set of possible outcomes of the derivation process. To avoid an unfair advantage for refuter (she may propose any set of states at the moment), we give prover the possibility to challenge and verify the prediction.

If prover challenges the prediction the proposition is verified by playing out the subplay i.e. the owner of X applies a rule $X \rightarrow_G \beta$ and we play until a position of shape $ww'\alpha$ is reached. If the state changes induced by w' are contained in the proposition, refuter wins the whole inclusion game. Otherwise, prover is declared winner. We argued above that we do not need the suffix α for this part of the play. Furthermore, if prover chooses to challenge the proposition, the whole game ends after a terminal word has been derived from X. Thus, in this case, we can discard the suffix α.

If prover accepts the proposition, she picks a terminal word w' in form of its state change from the set and the play carries on from $ww'\alpha$. The derivation process of X is skipped completely and in particular no grammar rule is applied to X.

The idea behind the propositions and challenges is the following. As in the summary approach, we can flatten the choices in the subplays to two subsequent choices, one for each player. The propositions reflect refuters influence in the subplays, as she can choose to include or exclude certain states from his prediction. Prover on her side fixes her choices in the subplay by picking a state from the prediction. Figure 14 depicts an example of a subplay and discusses different predictions.

$$\begin{tikzcd}
& & X_{\Box} \arrow[dl, Rightarrow] \arrow[dr, Rightarrow] & & \\
& aZ_{\bigcirc} \arrow[dl, Rightarrow] \arrow[d, Rightarrow] & & bZ_{\bigcirc} \arrow[d, Rightarrow] \arrow[dr, Rightarrow] & \\
ac & ad & & bc & bd
\end{tikzcd}$$

Figure 14.: Example of a derivation for non-terminal X. Note the following facts:
(1) Prediction $P = \{ac, ad\}$ would be losing for refuter ($\bigcirc$) if challenged, as prover owns X and can choose $X \Rightarrow b$.
(2) By choosing $P = \{ac, bc\}$, refuter chooses rule $Z \Rightarrow c$.
(3) If prover picks ac from P, she chooses rule $X \Rightarrow aZ$.

The idea to use predictions changes the course of the game. Plays do not only derive words, but also verify predictions. The initial proposition is the set of all states that are not final. Thus, the plays verify whether refuter can enforce the derivation (from S on) of a terminal word that induces a state change to a non-final state in A, i.e. to a word that is not accepted by A.

This approach ensures that we never need to store more symbols for the sentential forms than the length of the largest right-hand side of any grammar rule. Every time a rule is about to be applied at a sentential form $wX\alpha$, which only happens if prover challenges the prediction of refuter, the suffix α is discarded. If prover accepts the prediction, no rule is applied and the leftmost non-terminal X is deleted.

Figure 15 depicts how the two ideas from above lead to a finite graph. Vertex $\mathit{Check}(X\alpha, P_0, q_w)$

$$\begin{tikzcd}
& & & \mathit{Check}(X\alpha, P_0, q_w) \arrow[dl] \arrow[d, "\cdots"'] & \\
& & \mathit{Claim}(X\alpha, P', P_0, q_w) & \mathit{Claim}(X\alpha, P, P_0, q_w) \arrow[dll] \arrow[d, "\cdots"] \arrow[dr] & \\
& \mathit{Verify}(X, P, q_w) \arrow[dl] \arrow[d, "\cdots"'] & & \mathit{Check}(\alpha, P_0, q_{ww'}) & \mathit{Check}(\alpha, P_0, q_{ww''}) \\
\mathit{Check}(\beta_1, P, q_w) & \mathit{Check}(\beta_n, P, q_w) & & &
\end{tikzcd}$$

Figure 15.: Overview of the structure of the finite graph for the *GC game*.

represents sentential form $wX\alpha$ with prediction P_0. The terminal prefix w is represented in form of the state change q_w it induces on A from q_0 on. The node $\mathit{Check}(X\alpha, P_0, q_w)$ belongs to refuter. In the succeeding claim-nodes, the prediction for the subplay starting at $wX\alpha$ is made and there is such a node for each possible set of states P. The claim-nodes in turn belong to prover. Prover has two possibilities. She may move to the verify-node to challenge the prediction P. In this case, the owner of X has to pick a rule from $X \to_G \beta_1 \mid \cdots \mid \beta_n$ and the play continues by verifying the new prediction P. Alternatively, prover may accept the prediction. Then, she can pick a state $q_{ww'}$ from P, the subplay is skipped and the play continues with the old prediction P_0.

With the formal definition of the *GC game*, we concretize the above discussions into a finite graph. Finally, we show how to convert a winning strategy for the original game to one for the *GC game* and vice versa. This also proves that the winners coincide on both games and thus that the inclusion game can be solved by solving the *GC game*.

5.2. The *GC game*

In this section, we define the graph on which the game is played. The game graph of the *GC game* be given by $\mathcal{G}_{\mathrm{GC}}= (\mathcal{V}, \mathcal{E}, v_0)$, where $\mathcal{V} = \mathcal{V}_{\bigcirc} \dot{\cup} \mathcal{V}_{\Box}$ is the set of positions (divided between the players $\Box$ and $\bigcirc$), $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$ is the set of transitions and v_0 is the initial position.

The *GC game* is a reachability game on graph $\mathcal{G}_{\mathrm{GC}}$. Analogously to the graph game corresponding to the context-free game, refuter wins a play if it is maximal (the last position has no outgoing edge) and the last position is contained in some set $\mathcal{V}_{\mathrm{F}}$ that we specify below. Prover wins all other plays. Refuter wins the game if she has a strategy to enforce a play that is winning for her.

5.2.1. Positions of the *GC game*

We start by introducing the types of positions and their respective owners. We use the following notation. Sentential forms are represented by γ, γ', P, P' represent predictions and q, p represent states of A. Recall that we need a deterministic automaton on the right-hand side of the inclusion. If the automaton A given by the context-free game is non-deterministic, we first determinize it into an automaton A^{det}. Because we do not mention the (non-determinized) automaton A anymore, assume that the starting state of A^{det} is given by q_0, the set of states by Q, the transitions by $q \xrightarrow{a} p$ and the final states by Q_F. For reasons of comprehensibility, we postpone most of the explanations concerning the meaning of the positions until the definition of the transitions.

- *Check*(γ, P, q), owner$= \begin{cases} \text{prover , if } \gamma = \varepsilon \\ \text{prover, if } \gamma = a\gamma' \text{ for } a \in T \setminus \{\varepsilon\}, \gamma' \in \mathscr{S} \\ \text{refuter, else.} \end{cases}$

  The check-nodes represent sentential forms in the derivation process of shape $w\gamma\alpha$ where w is a terminal word s.t. $q_0 \xrightarrow{w} q$ and α is a sentential form. As the suffix α is discarded whenever a prediction is challenged, the node in general does not store the complete sentential form. In Figure 15 for example, the check-nodes *Check*(β_i, P, q_w) represent sentential forms $w\beta_i\alpha$.

  The purpose of the check-nodes is to track the target state in A^{det} of the terminal prefix w from q_0 on along the derivation process. This becomes more clear when we define the transitions.

  Intuitively, refuter wins the game starting from this node if the prediction P is correct for γ and q, i.e. she can enforce the derivation of a terminal word w from α s.t. $q \xrightarrow{w} p \in P$.

- *Verify*(X, P, q), owner = owner of X
  Verify nodes mark the beginning of a verify branch. In the next step, a rule $X \to_G \beta$ will be applied to X by its owner. Intuitively, refuter wins the game starting from this node if the prediction P is correct for X and q.Although the check- and verify-nodes seem very similar, we need to distinguish them as we only apply rules at verify-nodes.

- *Claim*$(X\gamma', P', P, q)$, owner =prover
  At the claim nodes, refuter makes his prediction P' about the subplays. Therefore, the claim-nodes belong to refuter. The old prediction P that has been inherited from the previous node has to be kept in case prover does not challenge the prediction. Intuitively, refuter wins from the claim-node if prediction P' is correct for X and q and if prediction P is correct for γ and for all $q' \in P'$.

- *Test*(q, P), owner= $\begin{cases} \text{refuter if } q \in P \\ \text{prover, else} \end{cases}$

  The test nodes are the end nodes of the game, i.e. they are the only nodes without a successor. They are winning for refuter if $q \in P$, i.e. the prediction is correct, and winning for prover otherwise. Thus, the set of winning nodes for refuter are $\mathcal{V}_\text{F} = \{\mathit{Test}(q, P) \in \mathcal{V} \mid q \in P\}$.

5.2.2. Edges of the *GC game*

Let us now define the edges of the graph:

- *Check*$(a\gamma', P, q) \longrightarrow$ *Check*(γ', P, p)
  If the first symbol of γ is a terminal, it will simply be consumed and its state change composed with q, i.e. $q \xrightarrow{a} p$. Note that this transition is unique as A^det is deterministic.

  If q tracked the state change of some w, p is the state change of wa. The owner of the check-node does not matter for this case as it has only one successor. To simplify the correctness proof, we choose prover to be the owner.

- *Check*$(\varepsilon, P, q) \longrightarrow$ *Test*(q, P)
  If $\gamma = \varepsilon$, the derivation process is finished and it remains to test whether the prediction is correct: $q \in P$? The owner of the check-node does not matter here as well and we choose prover again.

- *Check*$(X\gamma', P, q) \xrightarrow{\text{(for all predictions } P')}$ *Claim*$(X\gamma', P', P, q)$
  In this case, the check-node belongs to refuter. As we discussed above, refuter needs to make a prediction for the subplay. As refuter can predict any set of boxes, we need edges to all possible predictions and refuter is allowed to choose one.

- *Claim*$(X\gamma', P', P, q) \xrightarrow{\text{skip } (p \in P')}$ *Check*(γ', P, p)
  This edge leads into the skip-branch. The predecessor of the claim-node is the check-node *Check*$(X\gamma', P, q)$, representing a sentential form of shape $wX\gamma'\alpha$. State q of the check- and claim-node represents the target state of terminal prefix w in A^det (from q_0 on). If prover decides to skip the derivation process of X and accepts the prediction, she picks a state p from P'. This state represents the target state of the terminal prefix ww' of $ww'\gamma\alpha$, where w' has been derived from X. Therefore, we need such an edge for every $p \in P'$. The state p is stored in the check-node, which tracks the target state of the terminal prefix. This edge is called a skip-edge and the branch starting at the ckeck-node a skip-branch.

- *Claim*$(X\gamma', P', P, q) \xrightarrow{\text{(verify)}}$ *Verify*(X, P', q)
  This edge leads from the claim node, where the prediction for the subplay has been made, to the verify branch. If prover does not accept the prediction P', the subplay is played out. At this point, the information about the suffix γ' is discarded and a grammar rule can be applied in the next step. The prediction P is replaced by P'. We call this edge a verify-edge and the branch starting at the verify-node the verify-branch.

- $\mathit{Verify}(X, P, q) \xrightarrow{\text{(for all rules } X \Rightarrow \gamma)} \mathit{Check}(\gamma, P, q)$

  The verify-node belongs to the owner of X because he needs to choose the rule that will be applied. Thus, there exists such an edge from $\mathit{Verify}(X, P, q)$ to $\mathit{Check}(\gamma, P, q)$ for all rules $X \to_G \gamma$.

5.2.3. Starting node

As we consider the game from the point of view of refuter, the starting node of the game will be

$$c_0 = \mathit{Check}(S, P_{\text{rej}}, q_0), \text{ where } P_{\text{rej}} \text{ contains all non-accepting states.}$$

As already mentioned above, a play starting at a check-node of this shape is won by refuter if she can enforce the derivation of a terminal word w from S s.t. $q_0 \xrightarrow{w} q \in P_{\text{rej}}$. This means that we verify whether refuter can make sure that the derived word is not accepted by automaton A^{det}.

5.3. Determining the winner

After we constructed the finite game graph of the *GC game* from the grammar G and the determinized automaton A^{det}, it remains to determine the winner of the game and thereby also of the original inclusion game. As the game graph $\mathcal{G}_{\text{GC}}$ is finite, we can use the attractor $\mathsf{Attr}_{\bigcirc}(\mathcal{V}_{\text{F}}, \mathcal{E})$ for this task. It contains all nodes from which refuter can enforce a finite maximal play ending in a nodes in $\mathcal{V}_{\text{F}}$.

In case the parameters are unambiguous, we write $\mathsf{Attr}_{\bigcirc}^{(k)}$ for the fixed-point approximants and $\mathsf{Attr}_{\bigcirc}$ for the fixed-point. If the starting node $\mathit{Check}(S, P_{\text{rej}}, q_0)$ is contained in the attractor, refuter has a winning strategy for the *GC game* and thereby also in the inclusion game. If the starting node is not contained in the attractor, prover wins both games.

Instead of a simple attractor, we use an antichain-algorithm. Antichain algorithms have been introduced in [4]. The idea is to reduce the number of elements in the fixed-point approximants $\mathsf{Attr}_{\bigcirc}^{(k)}$ to speed up the attractor construction by deleting appropriate nodes from $\mathsf{Attr}_{\bigcirc}^{(k)}$. We make precise in a moment.

Formally, antichains are defined as follows.

Definition 17
An antichain of a partially ordered set $(M, \preceq)$ is a subset $M' \subseteq M$ s.t. all elements of M' are incomparable, i.e. $\forall e, e' \in M' : e \not\preceq e'$ and $e' \not\preceq e$.

We can combine the antichain idea with the attractor to reduce number of elements in the attractor. We exploit the structure of the *GC game*. Whenever refuter wins from a node $v = \mathit{Verify}(X, P, q)$, then she also wins from any node $v' = \mathit{Verify}(X, P', q)$, where $P \subseteq P'$. The reason for this is twofold. On one hand we have that the parts of the game graph reachable from v resp. v' are very similar in structure. The structure is mainly determined by the grammar rules and the states of A in form of the predictions. Thus, we have for each play $\pi_v = v, v_1, v_2, \ldots$ starting from v a corresponding play $\pi_{v'} = v', v'_1, v'_2, \ldots$ starting from v'. As long as no skip-edge is encountered on the plays, we have that v_i and v'_i are equal except for the predictions P resp. P'. As soon as a verify-node is reached, the predictions P, P' are discarded, replaced by the same prediction P'' and from this point on the nodes in the plays are exactly the same. In particular, this means that both plays either end in test-nodes $\mathit{Test}(q, P)$ resp. $\mathit{Test}(q, P')$ or in the same node $\mathit{Test}(q, P'')$ for some prediction P''.

On the other hand, we have that test-nodes only check for inclusion. Thus, if the last node in π_v belongs to refuter, the same holds for the last node in $\pi_{v'}$.

Intuitively, this is due to the fact that refuter wins from a node *Verify*(X, P, q) if she can enforce the derivation of some $w \in T^*$ from X s.t. $q \xrightarrow{w} p \in P$. If this is the case, it holds for the same word w that $q \xrightarrow{w} p \in P'$ as $P' \subseteq P$.

We take the set of nodes $\mathcal{V}$ of $\mathcal{G}_{\mathrm{GC}}$ as domain for the partially ordered set.

Definition 18
The order $\preceq$ is defined on the verify-nodes. We have that

$$\mathit{Verify}(X, P, q) \preceq \mathit{Verify}(X, P', q) \iff P \subseteq P'.$$

We will show that it is sufficient to only maintain $\preceq$-minimal verify-nodes in the attractor sets $\mathsf{Attr}_{\bigcirc}(\mathcal{V}_{\mathrm{F}}, \mathcal{E})^i$. Any other verify-nodes can be safely discarded. Therefore, we change the definition of the attractor to the following.

Definition 19
The antichain attractor is definied by

$$\begin{aligned}
&\mathsf{Attr}_{\mathbb{A},\bigcirc}(\mathcal{V}_F, \mathcal{E})^{(0)} = \mathcal{V}_F \\
&\mathsf{Attr}_{\mathbb{A},\bigcirc}(\mathcal{V}_F, \mathcal{E})^{(k+1)} = \mathbb{A}\Big(\mathsf{Attr}_{\mathbb{A},\bigcirc}(\mathcal{V}_F, \mathcal{E})^k \cup M_{new}^{(k+1)}\Big) \\
&\qquad \textit{where } M_{new}^{(k+1)} = \quad \{v \in \mathcal{V} \mid v \in \bigcirc \textit{ and } \exists\ (v, v') \in \mathcal{E} \textit{ with } v' \in \mathsf{Attr}_{\mathbb{A},\bigcirc}(\mathcal{V}_F, \mathcal{E})^i\} \\
&\qquad\qquad\qquad\qquad \cup \{v \in \mathcal{V} \mid v \in \Box \textit{ and } \forall\ (v, v') \in \mathcal{E} \textit{ have } v' \in \mathsf{Attr}_{\mathbb{A},\bigcirc}(\mathcal{V}_F, \mathcal{E})^i\} \\
&\qquad \textit{and } \mathbb{A}(M) = \{e \in M \mid \nexists e' \in M \textit{ s.t. } e' \preceq e\} \\
&\mathsf{Attr}_{\mathbb{A},\bigcirc}(\mathcal{V}_F, \mathcal{E}) = \bigcup_{k \in \mathbb{N}} \mathsf{Attr}_{\mathbb{A},\bigcirc}(\mathcal{V}_F, \mathcal{E})^{(k)}.
\end{aligned}$$

In case the parameters are unambiguous, we write $\mathsf{Attr}_{\mathbb{A},\bigcirc}^{(k)}$ and $\mathsf{Attr}_{\mathbb{A},\bigcirc}$ for the antichain attractor. Before we show that the antichain attractor correctly determines the winner of the $\mathcal{G}_{\mathrm{GC}}$, we state a few properties of the attractor sets $\mathsf{Attr}_{\bigcirc}^{(k)}$ and $\mathsf{Attr}_{\mathbb{A},\bigcirc}^{(k)}$. They follow directly from the definitions of the attractors.

Lemma 4
Let $v \in \mathcal{V}$. Then it holds that

1. - *Let $v \in \mathcal{V}_{\bigcirc}$: $v \in \mathsf{Attr}_{\bigcirc}^{(k)} \iff \exists\ (v, v') \in \mathcal{E}$ s.t. $v' \in \mathsf{Attr}_{\bigcirc}^{(k-1)}$*
   - *Let $v \in \mathcal{V}_{\Box}$: $v \in \mathsf{Attr}_{\bigcirc}^{(k)} \iff \forall\ (v, v') \in \mathcal{E}$ holds $v' \in \mathsf{Attr}_{\bigcirc}^{(k-1)}$*
2. *Suppose v is not a verify-node.*
   - *Let $v \in \mathcal{V}_{\bigcirc}$: $v \in \mathsf{Attr}_{\mathbb{A},\bigcirc}^{(k)} \iff \exists\ (v, v') \in \mathcal{E}$ s.t. $v' \in \mathsf{Attr}_{\mathbb{A},\bigcirc}^{(k-1)}$*
   - *Let $v \in \mathcal{V}_{\Box}$: $v \in \mathsf{Attr}_{\mathbb{A},\bigcirc}^{(k)} \iff \forall\ (v, v') \in \mathcal{E}$ holds $v' \in \mathsf{Attr}_{\mathbb{A},\bigcirc}^{(k-1)}$*
3. *Suppose now v is a verify node. Then, we get the following weaker claim (the verify-node may not be $\preceq$-minimal and thus may be discarded by the $\mathbb{A}$-operation).*
   - *Let $v \in \mathcal{V}_{\bigcirc}$: $v \in M_{new}^{(k)} \iff \exists\ (v, v') \in \mathcal{E}$ s.t. $v' \in \mathsf{Attr}_{\mathbb{A},\bigcirc}^{(k-1)}$*
   - *Let $v \in \mathcal{V}_{\Box}$: $v \in M_{new}^{(k)} \iff \forall\ (v, v') \in \mathcal{E}$ holds $v' \in \mathsf{Attr}_{\mathbb{A},\bigcirc}^{(k-1)}$*

From the discussion above we get the following Theorem.

Theorem 2
$\mathit{Check}(S, P_{rej}, q_0) \in \mathsf{Attr}_{\bigcirc} \iff \mathit{Check}(S, P_{rej}, q_0) \in \mathsf{Attr}_{\mathbb{A},\bigcirc}$.

Proof. "$\Leftarrow$" We can easily show by induction that $\mathsf{Attr}^{(k)}_{\mathbb{A},\bigcirc} \subseteq \mathsf{Attr}^{(k)}_{\bigcirc}$ for each $k \in \mathbb{N}$. Then the claim follows.

"$\Rightarrow$" Suppose that $\mathit{Check}(S, P_{\text{rej}}, q_0) \in \mathsf{Attr}_{\bigcirc}$. This means that at least one of the succeeding claim-nodes $\mathit{Claim}(S, P, P_{\text{rej}}, q_0)$ has to be contained in the attractor. Let now $P_{\min} \subseteq P$ be an inclusion-minimal prediction s.t. $\mathit{Claim}(S, P_{\min}, P_{\text{rej}}, q_0) \in \mathsf{Attr}_{\bigcirc}$.

This means on one hand that all succeeding check-nodes $\mathit{Check}(\varepsilon, P_{\text{rej}}, q)$ for $q \in P_{\min}$ are contained in the attractor. This means that the test-nodes succeeding these check-nodes are contained in $\mathsf{Attr}_{\bigcirc}$ as well and then by definition also in $\mathsf{Attr}_{\mathbb{A},\bigcirc}$. But then by Lemma 4, also the check-nodes $\mathit{Check}(\varepsilon, P_{\text{rej}}, q)$ are contained in $\mathsf{Attr}_{\mathbb{A},\bigcirc}$.

On the other hand, we can conclude from $\mathit{Claim}(S, P, P_{\text{rej}}, q_0) \in \mathsf{Attr}_{\bigcirc}$ that for the succeeding verify-node holds $v = \mathit{Verify}(S, P_{\min}, q_0) \in \mathsf{Attr}_{\mathbb{A},\bigcirc}$. But then we can deduce that the verify-node is $\preceq$-minimal in the attractor. Suppose the contrary, i.e. there is a verify-node $v' = \mathit{Verify}(S, P'_{\min}, q_0) \in \mathsf{Attr}_{\bigcirc}$ s.t. $v'' \preceq v'$. But then it would hold that claim-node $cl = \mathit{Claim}(()S, P_{\min'}, P_{\text{rej}}, q_0) \in \mathsf{Attr}_{\bigcirc}$. This is due to the fact that by $P'_{\min} \subseteq P_{\min}$, the set of check-nodes succeeding cl is a subset of the check-nodes succeeding $\mathit{Claim}(S, P, P_{\text{rej}}, q_0)$, which are all contained in $\mathsf{Attr}_{\bigcirc}$. But this contradicts the fact that we chose $P_{\min}$ minimal in the first place.

Thus, we can apply Lemma 5 and get that $\mathit{Verify}(S, P_{\min}, q_0) \in \mathsf{Attr}_{\mathbb{A},\bigcirc}$. As all the successors of the claim-node $\mathit{Claim}(S, P, P_{\text{rej}}, q_0)$ are contained in $\mathsf{Attr}_{\mathbb{A},\bigcirc}$, this also holds for the claim-node and thereby for the check-node $\mathit{Check}(S, P_{\text{rej}}, q_0)$.

□

We now prove Lemma 5 that we needed in the previous theorem.

Lemma 5
If $\mathit{Verify}(X, P, q) \in \mathsf{Attr}_{\bigcirc}$ and $\mathit{Verify}(X, P, q)$ is $\preceq$-minimal such, then also $\mathit{Verify}(X, P, q) \in \mathsf{Attr}_{\mathbb{A},\bigcirc}$.

Proof. We prove the following stronger claim. If $\mathit{Verify}(X, P, q) \in \mathsf{Attr}^{(k)}_{\bigcirc}$ and $\mathit{Verify}(X, P, q)$ is $\preceq$-minimal such, then also $\mathit{Verify}(X, P, q) \in \mathsf{Attr}^{(k)}_{\mathbb{A},\bigcirc}$. We prove this by induction over k.

Base case: $k = 0$
As there are no verify-nodes in $\mathsf{Attr}^{(0)}_{\bigcirc}$, the claim holds trivially.

Induction step: $k \to k + 1$
Let $v = \mathit{Verify}(X, P, q) \in \mathsf{Attr}^{(k+1)}_{\bigcirc}$ and $\mathit{Verify}(X, P, q)$ be $\preceq$-minimal. Then, by Lemma 4 (1.) either at least one or all successors of v have to be contained in $\mathsf{Attr}^{(k)}_{\bigcirc}$ depending on whether refuter or prover owns v. By construction of $\mathcal{G}_{\text{GC}}$, all successors are of shape $\mathit{Check}(\alpha, P, q)$, where α is the right-hand side of a grammar rule $X \to_G \alpha$. We show in Lemma 6 that if such a successor node is contained in $\mathsf{Attr}^{(k)}_{\bigcirc}$, then it is also contained in $\mathsf{Attr}^{(k)}_{\mathbb{A},\bigcirc}$. Thus, by Lemma 4 (3.) we can conclude that $v = \mathit{Verify}(X, P, q) \in M^{(k+1)}_{\text{new}}$.

It remains to show that v is $\preceq$-minimal such. But if we assume the contrary, namely that there exists some $\preceq$-minimal node $v' = \mathit{Verify}(X, P', q) \in M^{(k+1)}_{\text{new}}$ with $P' \subsetneq P$. Then also $v' \in \mathsf{Attr}^{(k+1)}_{\mathbb{A},\bigcirc}$. As we have $\mathsf{Attr}^{(k+1)}_{\mathbb{A},\bigcirc} \subseteq \mathsf{Attr}^{(k+1)}_{\bigcirc}$, v' is contained in $\mathsf{Attr}^{(k+1)}_{\bigcirc}$ contradicting the fact that v was chosen $\preceq$-minimal. Finally, we can deduce that $v \in \mathsf{Attr}^{(k+1)}_{\mathbb{A},\bigcirc}$.

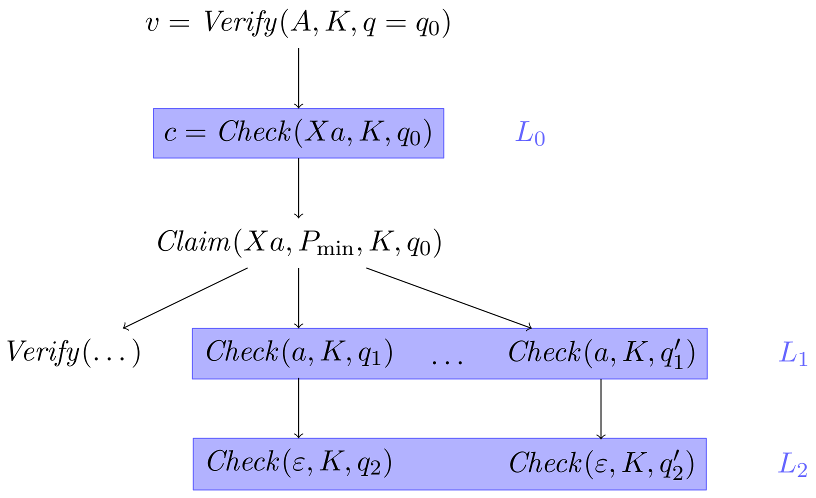
□

To finalize the proof of Lemma 5 and Theorem 2 it remains to show the following claim.

Lemma 6
Let $c = \mathit{Check}(\alpha, P, q)$ be a successor node of v from above. If $c \in \mathsf{Attr}^{(k)}_{\bigcirc}$, then also $c \in \mathsf{Attr}^{(k)}_{\mathbb{A},\bigcirc}$.

Proof. For this proof, we use the concept of layers $L_0, \dots, L_n$. In layer L_j, we collect check-nodes of shape $\mathit{Check}(\alpha', P, q')$ where α' is a suffix of α. Furthermore, the nodes in each layer have the following properties.

- They are contained in $\mathsf{Attr}^{(k)}_{\bigcirc}$.
- They are reachable from c via a path containing no verify-node.
- The paths only contain claim-nodes that are inclusion minimal in $\mathsf{Attr}^{(k)}_{\bigcirc}$ i.e. nodes $\mathit{Claim}(\alpha', P', P, q')$ s.t. no claim-node $\mathit{Claim}(\alpha', P'', P, q')$ with $P'' \subsetneq P'$ is contained in the attractor.



$v = \mathit{Verify}(A, K, q = q_0)$
$c = \mathit{Check}(Xa, K, q_0)$
L_0
$\mathit{Claim}(Xa, P_{\min}, K, q_0)$
$\mathit{Verify}(\dots)$
$\mathit{Check}(a, K, q_1)$ ... $\mathit{Check}(a, K, q_1')$
L_1
$\mathit{Check}(\varepsilon, K, q_2)$
$\mathit{Check}(\varepsilon, K, q_2')$
L_2

Figure 16.: Layer construction starting at c (all depicted nodes are contained in $\mathsf{Attr}^{(k)}_{\bigcirc}$).

We start by $L_0 = \{c\}$ and collect the nodes in L_{j+1} by taking appropriate successors of the nodes from L_j. Figure 16 depicts an example of this layer construction. The goal is to show that $L_0 \subseteq \mathsf{Attr}^{(k)}_{\mathbb{A},\bigcirc}$, then the claim follows.

Intuitively, this is the case because we only chose inclusion minimal claim-nodes in the paths. We do not need to consider paths containing verify-nodes as we can use the induction hypothesis to prove that they are contained in $\mathsf{Attr}^{(k)}_{\mathbb{A},\bigcirc}$.

Therefore, we show that $L_j \subseteq \mathsf{Attr}^{(k-f(j))}_{\mathbb{A},\bigcirc} \subseteq \mathsf{Attr}^{(k)}_{\mathbb{A},\bigcirc}$ for an adequate function $f : \mathbb{N} \to \mathbb{N}$, $f(j) \geq 0$ and each layer L_j. Although it seems intuitive at first to define $f(j) = j$ for all j, it does not guarantee that $L_j \subseteq \mathsf{Attr}^{(k-f(j))}_{\mathbb{A},\bigcirc}$ holds. The reason is the following. Suppose the check-nodes of layer L_j are of shape $\mathit{Check}(X\alpha', P, q')$ for some suffix $X\alpha'$ of α and that $L_j \subseteq \mathsf{Attr}^{(k-j)}_{\mathbb{A},\bigcirc}$. To collect the nodes for layer L_{j+1}, we have to skip the direct successors of the check-nodes as they are claim nodes. Thus, we would have $L_{j+1} \subseteq \mathsf{Attr}^{(k-j-2)}_{\mathbb{A},\bigcirc}$ and the index of the attractor does not match the index of the layer anymore. For example in Figure 16 this is the case for L_1.

This case always arises whenever α' starts with a non-terminal. Furthermore, we have to take into account how often this happened for the previous layers. Thus, we define

$$f(j) = j + \#N(\alpha, j)$$

where $\#N(\alpha, j)$ equals the sum of the non-terminals in $\alpha_1, \dots, \alpha_j$.

Let now $|\alpha| = n$. During the layer construction, we maintain the following three invariants.

1. The elements of L_j are contained in the attractor: $L_j \subseteq \mathsf{Attr}^{(k-f(j))}_{\bigcirc}$.

2. For $\mathit{Check}(\alpha', P, q') \in L_j$ holds that $|\alpha'| = n - j$.

   We need this invariant to ensure that the layer construction eventually terminates.

3. If $L_j \subseteq \mathsf{Attr}^{k-f(j)}_{\mathbb{A},\bigcirc}$, then also $L_{j-1} \subseteq \mathsf{Attr}^{k-f(j-1)}_{\mathbb{A},\bigcirc}$ for $j \geq 1$.

   From this invariant we will deduce $L_0 \subseteq \mathsf{Attr}^{(k)}_{\mathbb{A},\bigcirc}$ by showing that $L_n \subseteq \mathsf{Attr}^{k-f(n)}_{\mathbb{A},\bigcirc}$ at the end.

The first layer only contains node c, $L_0 = \{c\}$. The first invariant is met as $k - f(0) = k$ and the second invariants holds trivially. The third one does not apply.

Suppose now that we already constructed L_j and want to derive L_{j+1}. Let $c' = \mathit{Check}(\alpha', P, q') \in L_j$. We have to distinguish two cases.

First, let $\alpha' = a\beta$ for $a \in T$ and $\beta \in \mathscr{S}^*$. Then, c' only has one successor c''(see Figure 17). Then, we include c'' in L_{j+1}. The invariants are all satisfied:

Case 1: $\alpha' = a\beta$

- As $c' \in L_j$, $c' \in \mathsf{Attr}^{(k-f(j))}_{\bigcirc}$ by the first invariant of L_j. But then, by Lemma 4 (1.), $c'' \in \mathsf{Attr}^{(k-f(j))-1)}_{\bigcirc}$. As the first symbol of α' is a terminal, we have that $f(j+1) = f(j) + 1$, thus the first invariant is satisfied for L_{j+1}.
- From the second invariant of L_j, we get that $|a\beta| = n - j$. Thus $|\beta| = n - j - 1 = n - (j+1)$.
- Finally, if $c'' \in \mathsf{Attr}^{(k-f(j+1))}_{\mathbb{A},\bigcirc}$, $c' \in \mathsf{Attr}^{(k-f(j+1)+1)}_{\mathbb{A},\bigcirc} = \mathsf{Attr}^{(k-f(j))}_{\mathbb{A},\bigcirc}$ by Lemma 4 (2.).

$$\begin{tikzcd}
c' = \mathit{Check}(a\beta, P, q') \arrow[d] \\
c'' = \mathit{Check}(\beta, P, q'')
\end{tikzcd}$$

Figure 17.: Case $\alpha' = a\beta$.

Case 2: $\alpha' = X\beta$

Second, let $\alpha' = X\beta$ for $X \in N$. Then, the successors of c' are claim-nodes of shape $\mathit{Claim}(X\beta, P', P, q)$. As $c' \in L_j$, $c' \in \mathsf{Attr}^{(k-f(j))}_{\bigcirc}$ by the first invariant of L_j. Thus, at least one of the succeeding claim-nodes is part of the attractor $\mathsf{Attr}^{(k-f(j)-1)}_{\bigcirc}$ (Lemma 4 (1.)). We consider one of the claim-nodes cl with inclusion minimal prediction $P' = P_{\min}$ among those. Figure 18 depicts the relevant part of $\mathcal{G}_{\mathrm{GC}}$.

$$\begin{tikzcd}
 & & c' = \mathit{Check}(X\beta, P, q') \arrow[dll] \arrow[d] \arrow[d, phantom, shift left=4, "\cdots"] & \\
\mathit{Claim}(\dots) & & \mathit{cl} = \mathit{Claim}(X\beta, P_{\min}, P, q') \arrow[dll] \arrow[d] \arrow[d, phantom, shift left=4, "\cdots"] \arrow[dr] & \\
v' = \mathit{Verify}(X, P_{\min}, q') & & \mathit{Check}(\dots) & c'' = \mathit{Check}(\beta, P, q'')
\end{tikzcd}$$

$$\underbrace{\mathit{Check}(\dots) \quad c'' = \mathit{Check}(\beta, P, q'')}_{\text{one check-node for all } q'' \in P_{\min}}$$

Figure 18.: Case $\alpha' = X\beta$.

We include all check-nodes c'' succeeding $\mathit{cl} = \mathit{Claim}(X\beta, P_{\min}, P, q')$ in L_{j+1}. This satisfies all the invariants:

- As cl is contained in $\mathsf{Attr}_{\bigcirc}^{(k-f(j)-1)}$, for all of the nodes c'' must hold $c'' \in \mathsf{Attr}_{\bigcirc}^{(k-f(j)-2)}$. As the first symbol of α' is a non-terminal this time, we have $f(j+1) = j(j)+2$. Thus, we can conclude that $c'' \in \mathsf{Attr}_{\bigcirc}^{(k-f(j+1))}$.
- The second invariant holds as $|X\beta| = n - j$ and therefore $|\beta| = n - j + 1$.
- The third invariant also holds. As $cl \in \mathsf{Attr}_{\bigcirc}^{(k-f(j)-1)}$, its succeeding verify-node $v' = \mathit{Verify}(X, P_{\min}, q')$ is contained in $\mathsf{Attr}_{\bigcirc}^{(k-f(j)-2)} = \mathsf{Attr}_{\bigcirc}^{(k-f(j+1))}$ by Lemma 4 (1.). We can show that because we chose P_{min} to be inclusion minimal, the verify-node v' is $\preceq$-minimal in $\mathsf{Attr}_{\bigcirc}^{(k-f(j+1))}$.

  Suppose the contrary, i.e. there is a verify-node $v'' = \mathit{Verify}(X, P'_{\min}, q') \in \mathsf{Attr}_{\bigcirc}^{(k-f(j+1))}$ s.t. $v'' \preceq v'$. But then it would hold that claim-node $cl'' = \mathit{Claim}(X\beta, P_{\min}\text{'}, P, q') \in \mathsf{Attr}_{\bigcirc}^{(k-f(j)-1)}$. This is due to the fact that by $P'_{\min} \subseteq P_{\min}$, the set of check-nodes succeeding cl'' is a subset of the check-nodes succeeding cl, which are all contained in $\mathsf{Attr}_{\bigcirc}^{(k-f(j+1))}$. But this contradicts the fact that we chose $P_{\min}$ minimal in the first place.

  Thus, we can apply the induction hypothesis of Lemma 5 and get that $v' \in \mathsf{Attr}_{\mathbb{A},\bigcirc}^{k-f(j+1)}$. Thus, if $L_{j+1} \subseteq \mathsf{Attr}_{\mathbb{A},\bigcirc}^{k-f(j+1)}$, then also $c' \in \mathsf{Attr}_{\mathbb{A},\bigcirc}^{k-f(j)}$ (apply Lemma 4 (2.) twice).

Using invariant 2, we can conclude that there are only finitely many layers, more precisely $L_0, \ldots, L_{n+1}$ ($n = |\alpha|$). The check-nodes of the last layer are all of shape $\mathit{Check}(\varepsilon, P, q')$, whose successors are test-nodes. Thus, no further layer L_{n+2} can be constructed. Let $\mathcal{M}_P$ be the set of the corresponding test-nodes.

From the first invariant, we get that $L_{n+1} \in \mathsf{Attr}_{\bigcirc}^{(k-f(n))}$ and thus, by Lemma 4 (1.) $\mathcal{M}_P \subseteq \mathsf{Attr}_{\bigcirc}^{(k-f(n+1)-1)}$. From this, we can also deduce that $(k - f(n+1) - 1) = 0$. By definition of the attractor, test-nodes can only be contained in the attractor $\mathsf{Attr}_{\mathbb{A},\bigcirc}$ if they are winning i.e. $\mathcal{M}_P \subseteq \mathcal{V}_{\mathrm{F}}$. But then, $\mathcal{M}_P \subseteq \mathsf{Attr}_{\mathbb{A},\bigcirc}^{(0)}$ and thus, by Lemma 4 (2.) $L_{n+1} \subseteq \mathsf{Attr}_{\mathbb{A},\bigcirc}^{(1)} = \mathsf{Attr}_{\mathbb{A},\bigcirc}^{(k-f(n+1))}$.

Finally, we can inductively conclude by the third invariant that $L_0 \subseteq \mathsf{Attr}_{\mathbb{A},\bigcirc}^{(k-f(0))} = \mathsf{Attr}_{\mathbb{A},\bigcirc}^{(k)}$ as desired.

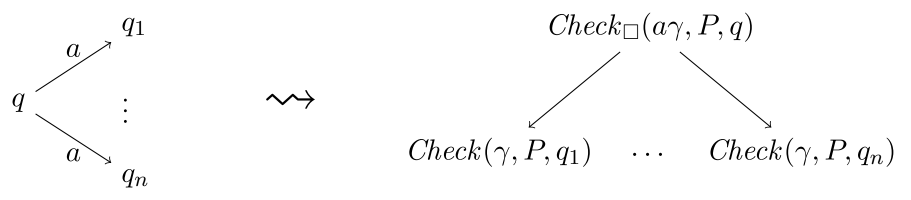
□

5.4. Non-determinism

As we already hinted in the beginning of this section, the *Guess & Check approach* can only be applied if we have a deterministic automaton on the right-hand side of the inclusion.

The following example shows that we can not adapt the *GC game* to work with a non-deterministic automaton s.t. the winner of the original game and the *GC game* coincide. To deal with the non-determinism, we have to let one of the players resolve the non-deterministic choices. Obviously, we can not let refuter handle this task, as she would choose non-accepting paths in A even if accepting ones exist and the word should be accepted. Thus, we assign the task to prover by applying the following modifications to the game graph $\mathcal{G}_{\mathrm{GC}}$ (see Figure 19).

As we do not have a deterministic automaton on the right-hand side of the inclusion, check-nodes of shape $\mathit{Check}(a\gamma, P, q)$ may now have several successors instead of only one. More precisely, the successors are all check-nodes $\mathit{Check}(\gamma, P, q_i)$ s.t. there is a transition $q \xrightarrow{a} q_i$ in A. The check-nodes of shape $\mathit{Check}(a\gamma, P, q)$ (with a terminal a at the beginning of the sentential form) belong to prover, as she is allowed to resolve the non-determinism. Otherwise, the $\mathcal{G}_{\mathrm{GC}}$ remains as defined above.



q
a
q_1
$\vdots$
a
q_n
$\rightsquigarrow$
$\mathit{Check}_{\Box}(a\gamma, P, q)$
$\mathit{Check}(\gamma, P, q_1)$
$\cdots$
$\mathit{Check}(\gamma, P, q_n)$

Figure 19.: Modification of the $\mathcal{G}_{\text{GC}}$. Prover resolves the non-determinism.

Consider now the following example of a context-free game given by grammar G and automaton A (Figure 20).

$$G: \begin{array}{l} S_{\bigcirc} \rightarrow_G aX \\ X_{\bigcirc} \rightarrow_G b \mid c \end{array}$$

A :

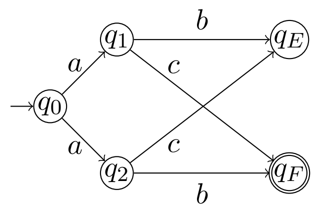


Figure 20.: The *GC game* does not work with non-deterministic automata.

The example is actually the same as the example in Section 4 showing that the saturation method does not work with non-deterministic automata as well. This is not surprising as the reason why the non-determinism is an issue is the same for both approaches.

Figure 21 shows a part of the game graph $\mathcal{G}_{\text{GC}}$. As the game graphs in the *GC game* tend to become very large, we only represent a snippet of the full game graph. The omitted parts are hinted in gray. The depicted snippet of the game graph mainly covers the parts of $\mathcal{G}_{\text{GC}}$ that are contained in the attractor $\mathsf{Attr}_{\mathbb{A},\bigcirc}(\mathcal{V}_{\text{F}}, \mathcal{E})$. The nodes that are contained in the attractor are marked in red. In particular, the attractor contains the starting node $\mathit{Check}(S, P_{\text{rej}}, q_0)$, indicating that refuter wins both the *GC game* and the context-free game. But obviously, prover should win the context-free game as all derivable words from G are accepted by A.

The reason for this mismatch is the same as for the saturation approach. By letting prover resolve the non-determinism, she has to fulfill a harder task to win the game. She has to show that the automaton A can simulate every branching behavior of the $\mathcal{G}_{\text{GC}}$ by an accepting path in A, which is a strictly stronger property than language inclusion.

Thus, we are obliged to determinize the finite automaton A into a deterministic automaton A^{det} and use the latter to construct the $\mathcal{G}_{\text{GC}}$.

5.5. Correctness of the Guess & Check approach.

Now that we have defined the *GC game*, it remains to show that the winning strategy for the inclusion game can be converted to a winning strategy for the *GC game* and vice versa. This also proves that the winner of both games coincides.

Theorem 3
If refuter has a winning strategy for the context-free game $\mathcal{G}$, then there also exists one for $\mathcal{G}_{GC}$ and vice versa.

Proof. We show this by converting a winning strategy for $\mathcal{G}$ to one for the $\mathcal{G}_{\text{GC}}$ and vice versa. This is elaborated in sections 5.5.1 and 5.5.2. □

The proof is an adapted version of Lemma 5.1 in the lecture notes [11].

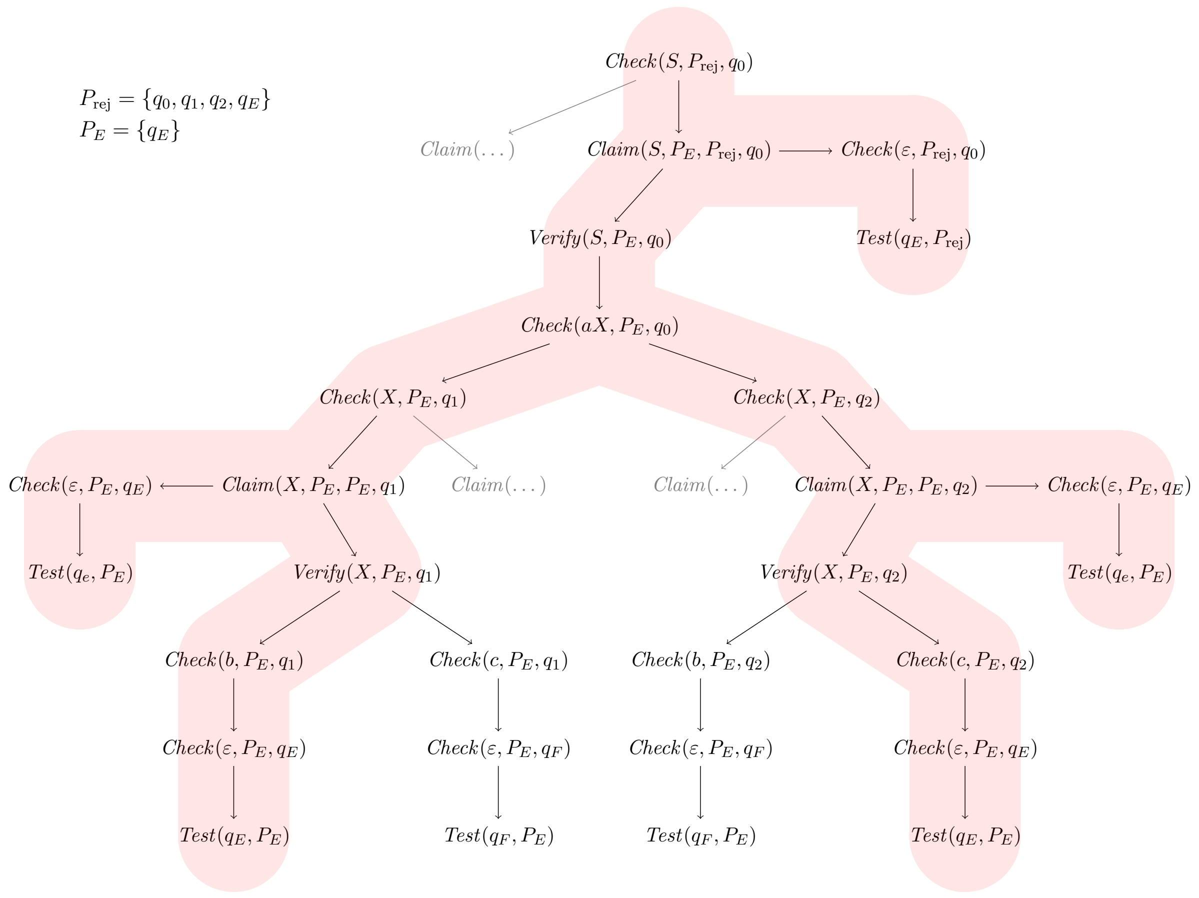

$P_{\mathrm{rej}} = \{q_0, q_1, q_2, q_E\}$
$P_E = \{q_E\}$
$\mathit{Check}(S, P_{\mathrm{rej}}, q_0)$
$\mathit{Claim}(\dots)$
$\mathit{Claim}(S, P_E, P_{\mathrm{rej}}, q_0)$
$\mathit{Check}(\varepsilon, P_{\mathrm{rej}}, q_0)$
$\mathit{Verify}(S, P_E, q_0)$
$\mathit{Test}(q_E, P_{\mathrm{rej}})$
$\mathit{Check}(aX, P_E, q_0)$
$\mathit{Check}(X, P_E, q_1)$
$\mathit{Check}(X, P_E, q_2)$
$\mathit{Check}(\varepsilon, P_E, q_E)$
$\mathit{Claim}(X, P_E, P_E, q_1)$
$\mathit{Claim}(\dots)$
$\mathit{Claim}(\dots)$
$\mathit{Claim}(X, P_E, P_E, q_2)$
$\mathit{Check}(\varepsilon, P_E, q_E)$
$\mathit{Test}(q_e, P_E)$
$\mathit{Verify}(X, P_E, q_1)$
$\mathit{Verify}(X, P_E, q_2)$
$\mathit{Test}(q_e, P_E)$
$\mathit{Check}(b, P_E, q_1)$
$\mathit{Check}(c, P_E, q_1)$
$\mathit{Check}(b, P_E, q_2)$
$\mathit{Check}(c, P_E, q_2)$
$\mathit{Check}(\varepsilon, P_E, q_E)$
$\mathit{Check}(\varepsilon, P_E, q_F)$
$\mathit{Check}(\varepsilon, P_E, q_F)$
$\mathit{Check}(\varepsilon, P_E, q_E)$
$\mathit{Test}(q_E, P_E)$
$\mathit{Test}(q_F, P_E)$
$\mathit{Test}(q_F, P_E)$
$\mathit{Test}(q_E, P_E)$

Figure 21.: Part of the game graph $\mathcal{G}_{\mathrm{GC}}$ for the context-free game in Figure 20. The nodes contained in the attractor $\mathsf{Attr}_{\mathbb{A},\bigcirc}(\mathcal{V}_{\mathrm{F}}, \mathcal{E})$ are marked with red background.

5.5.1. From the inclusion game to the *GC game*.

Let us assume that refuter wins the inclusion game $\mathcal{G}$ from the initial position S. If there exists a winning strategy for refuter from S for the context-free game, there also exists a positional winning strategy σ. This result has been shown in [5]. We use σ to construct a strategy σ' (not necessarily positional) for refuter in $\mathcal{G}_{\mathrm{GC}}$ from the initial position $\mathit{Check}(S, P_{\mathrm{rej}}, \rho_\varepsilon)$ and then show that σ' is indeed a winning strategy.

We construct the strategy inductively. In the induction step, we consider the play prefixes which are conform to the strategy σ' that has been constructed so far and update σ' to prolong them. For the sake of simplicity, all the play prefixes ω in $\mathcal{G}_{\mathrm{GC}}$ which are mentioned in the following construction are conform to σ'. We define for each play prefix ω in the induction step the following two (partial) functions.

- strategy σ': prefix in $\mathcal{G}_{\mathrm{GC}} \mapsto$ position in $\mathcal{G}_{\mathrm{GC}}$
  The strategy σ' takes a play prefix ω in $\mathcal{G}_{\mathrm{GC}}$ as input and returns a successor of $\mathit{last}(\omega)$, i.e. the last position in ω.
- function f : prefix in $\mathcal{G}_{\mathrm{GC}}$ ending in check node $\mapsto$ prefix in $\mathcal{G}$
  Function f maps a prefix ω in $\mathcal{G}_{\mathrm{GC}}$ to a prefix π in $\mathcal{G}$. We further require that f is prefix faithful i.e. if $\omega_1 \sqsubseteq \omega_2$, then $f(\omega_1) \sqsubseteq f(\omega_2)$.

The idea of the construction is the following: Function f is used to simulate play prefixes ω in $\mathcal{G}_{\mathrm{GC}}$ by play prefixes π in $\mathcal{G}$. For the simulated prefix π, we can use σ to determine a successor position in $\mathcal{G}$, which in turn will be used to update σ' to deal with ω in $\mathcal{G}_{\mathrm{GC}}$. Both f and σ' need to be updated in each induction step. Function f depends on the play prefix constructed so far by σ' and σ' on the simulated play prefix in $\mathcal{G}$.

For both f and σ', we require the following invariants to hold along the induction:

1. If play prefix ω in $\mathcal{G}_{\text{GC}}$ ends in $\mathit{Check}(\gamma, P, q)$, then $f(\omega) = \pi, v\gamma\alpha$, where π is a play prefix in $\mathcal{G}$, v is a terminal word s.t. $q_0 \xrightarrow{v} q$ and α some sentential form. This invariant is required to ensure that $\pi = f(\omega)$ reflects ω in $\mathcal{G}$ and that the step $(\mathit{last}(\pi), \sigma(\mathit{last}(\pi)))$ can be simulated from $\mathit{last}(\omega)$ in $\mathcal{G}_{\text{GC}}$.

2. The play prefixes $f(\pi)$ defined in the induction step must be valid plays in $\mathcal{G}$ and conform to σ. We need this invariant to prove that the constructed strategy σ' is a winning strategy by reasoning about the simulated plays in $\mathcal{G}$.

3. Let

$$\omega = \omega_1, \underbrace{\mathit{Check}(\gamma, P, q), \omega_2, \mathit{Check}(\beta, P', q')}_{\omega_3}$$

   be a play prefix in $\mathcal{G}_{\text{GC}}$ and there is no verify-node in ω_3. Then we have the following property.

$$f(\omega_1, \mathit{Check}(\gamma, P, q)) = \pi_1, v\gamma\alpha$$
$$f(\omega) = \pi_1, v\gamma\alpha, \pi_2, vw\beta\alpha$$

   We need this invariant to be able to reason about the suffix α of the last position $vw\beta\alpha = \mathit{last}(f(\omega))$. The terminal prefix vw and sentential form β can be read off from $\mathit{Check}(\beta, P', q') = \mathit{last}(\omega)$. But because the suffix γ' in a node $\mathit{Claim}(X\gamma', P', P, q)$ is discarded if the verify-branch is entered during the play, no information about the suffix can be extracted from the check-nodes. By invariant 1, we only get that $\mathit{last}(f(\omega_1, \mathit{Check}(\gamma, P, q))) = v\gamma\alpha$ and $\mathit{last}(f(\omega)) = v'\beta\alpha'$, without any relation between α and α'. But, this is not sufficient to prove that the play ω is winning. However, if no verify node is located between $\mathit{Check}(\gamma, P, q)$ and $\mathit{Check}(\beta, P', q')$ in ω, we can conclude that $\alpha = \alpha'$. Intuitively, this is the case because no part of γ is discarded during the play ω_3.

   We only prove this for play prefixes of ω ending in consecutive check-nodes i.e. there is no check-node in ω_2. But this can easily be generalized to prefixes with non-consecutive check-nodes by induction.

We first define f for the prefixes in $\mathcal{G}_{\text{GC}}$ of length l. As f is only defined for prefixes ending on a check-node, we only need to consider such play prefixes ω. Afterwards, we will define $\sigma'(\omega)$ based on the simulated play $f(\omega)$.

Base case: $\mid \omega \mid = 1$
Then $\omega = \mathit{Check}(S, P_{\text{rej}}, q_0)$. We set $f(\omega) = S$, which satisfies the first two invariants, the third does not apply.

Induction step: $\mid \omega \mid \geq 2$

1. $\omega = \omega', \mathit{Check}(a\gamma, P, q), \mathit{Check}(\gamma, P, q')$, where $q \xrightarrow{a} q'$.
   From the invariants, we know that $f(\omega' \mathit{Check}(a\gamma, P, q)) = \pi, va\gamma\alpha$, which is conform to σ. We define

$$f(\omega) = \pi, va\gamma\alpha = f(\omega', \mathit{Check}(a\gamma, P, q)).$$

   This trivially satisfies all invariants.

Note that $f(\omega_1) = f(\omega_2)$ can only happen for finitely many $\omega_1 \sqsubseteq \omega_2$, more precisely at most until the whole sentential form γ has been consumed. This insight is important for the correctness proof.

2. $\omega = \omega'$, $\mathit{Check}(X\gamma, P, q)$, $\mathit{Claim}(X\gamma, P', P, q)$, $\mathit{Verify}(X, P', q)$, $\mathit{Check}(\beta, P', q)$
In this case, prover challenged the prediction P' and entered the subplay. Again by the invariants, we can assume that $f(\omega' \mathit{Check}(X\gamma, P, q)) = \pi, vX\gamma\alpha$ and this is a play conform to σ. We set
$$f(\omega) = \pi, vX\gamma\alpha, v\beta\gamma\alpha.$$
This satsfies the first invariant. By construction of the game $\mathcal{G}_{\mathrm{GC}}$, we know that there exists a rule $X \rightarrow_G \beta$. Otherwise, the edge $(\mathit{Verify}(X, P', q), \mathit{Check}(\beta, P', q))$ would not exist. Therefore, the second invariant is satisfied as well. The third invariant does not apply.

3. $\omega = \omega'$, $\mathit{Check}(X\gamma, P, q)$, $\mathit{Claim}(X\gamma, P', P, q)$, $\mathit{Check}(\gamma, P, q')$, where $q' \in P'$.
Here, we consider the case where prover accepted the prediction and picked state $q' \in P'$. By the invariants, construction and the definition of σ' (see below), we can assume the following:
   - $f(\omega' \mathit{Check}(X, P, q)) = \pi, vX\gamma\alpha$, which is conform to σ.
   - $q' \in P'$
   - For all $q' \in P'$ there exists a valid play $vX\gamma\alpha, n_1, \ldots, n_k, vw\gamma\alpha$ conform to σ s.t. $q \xrightarrow{w} q'$. In this play, w is derived from X.

   Using these assumptions, we define
$$f(\omega) = \pi, vX\gamma\alpha, n_1, \ldots, n_k, vw\gamma\alpha$$
   which satisfies our invariants.

This concludes the definition of f. We now turn our attention to σ' for the prefixes ω of length l. As the claim- and jump-nodes belong to prover and the test-nodes do not have a successor, we only need to consider play prefixes ending with check-nodes of shape $\mathit{Check}(X\gamma, P, q)$ and verify-nodes where the non-terminal belongs to refuter. Let $f(\omega) = \pi, v\gamma\alpha$ in the following.

A. $\omega = \omega'$, $\mathit{Verify}(X, P, q)$, where $X \in \bigcirc$
In the next step, refuter needs to choose one of the check-nodes $\mathit{Check}(\beta, P, q)$. Intuitively, this move would correspond to the choice of the rule $X \rightarrow_G \beta$. We want to mimic the choice of σ on the simulated play $f(\omega)$. Using operator $r_\sigma()$, we can extract the grammar rule used in a step $(\gamma, \sigma(\gamma))$ of a play in $\mathcal{G}$.

Definition 20
$r_\sigma(vX\alpha) = \beta$, if $\sigma(vX\alpha) = v\beta\alpha$ and X belongs to refuter

We fix
$$\sigma'(\omega) = \mathit{Check}(r_\sigma(S), P, q).$$

B. $\omega = \omega'$, $\mathit{Check}(X\gamma, P, q)$
By his next move, refuter has to make a prediction about the outcome of the subplay. Refuter has to ensure that she wins both the verify- and the skip-branch. The prediction has to include all possible state changes of terminal words derivable from X according

to the winning strategy σ. Let thus P' contain all q' s.t. $q \xrightarrow{w} q'$ and there exists a valid play from $vX\gamma\alpha$ to some sentential form $vw\gamma\alpha$ that is conform to σ. We set

$$\sigma'(\omega) = \mathit{Claim}(X\gamma, P', P, q).$$

Now that we have constructed a strategy σ' for refuter, it remains to prove that it is indeed a winning strategy. This means that any play starting from $\mathit{Check}(S, P_{\mathrm{rej}}, q_0)$ and conform to σ' ends in a test node $\mathit{Test}(p, P)$ with $p \in P$. Let now ω be any play starting at $\mathit{Check}(S, P_{\mathrm{rej}}, q_0)$ and conform to σ'.

We first prove that ω eventually visits a test-node, meaning it can not be infinite.

Lemma 7
Let ω be a play in $\mathcal{G}_{GC}$ starting from $\mathit{Check}(S, P_{rej}, q_0)$ and conform to σ'. Then ω is finite and ends with a test-node.

Proof. Towards a contradiction, assume that ω never visits a test-node. By construction of $\mathcal{G}_{\mathrm{GC}}$ this means that we visit infinitely many check-nodes. Consider now $f(\omega)$. As $|f(\omega_1)| = |f(\omega_2)|$ can only happen for finitely many prefixes ω_1, ω_2 of ω and for all other prefixes $\omega_3 \sqsubseteq \omega_4$ we have $|f(\omega_3)| < |f(\omega_4)|$, $f(\omega)$ must be of infinite length too. But by the second invariant $f(\omega)$ is a valid play conform to σ. As infinite plays are losing for refuter we have a contradiction. Thus, ω must be of finite length. Because the test-nodes are the only nodes without successor, ω must end with a test-node. $\square$

Finally, we show that the test-node that concludes ω belongs to refuter.

Lemma 8
Play ω ends with a test node $\mathit{Test}(q, P)$ s.t. $q \in P$.

Proof. We distinguish between two cases.

First, we assume that there is no verify edge in ω. Figure 22 (left-hand side) illustrates this case. As there is no verify edge in ω, the initial prediction P_{rej} is never replaced. Therefore, ω ends in a test-node with prediction P_{rej}. We show that state q is included in P_{rej} using the first and third invariant. Let ω_1 be the prefix of ω containing only the initial position $\mathit{Check}(S, P_{\mathrm{rej}}, q_0)$ and ω_2 be the prefix obtained by removing the last node $\mathit{Test}(q, P_{\mathrm{rej}})$.

By construction, $\mathit{last}(f(\omega_1)) = f(\mathit{Check}(S, P_{\mathrm{rej}}, q_0)) = S = vS\alpha$ for $v = \alpha = \varepsilon$. Using the first invariant, we get that $\mathit{last}(f(\omega_2)) = vw\varepsilon\alpha'$ s.t. $q_0 \xrightarrow{w} q$. By the third invariant, we get that $v = \alpha = \varepsilon$. Finally, we can use the second invariant which guarantees us that $f(\omega)$ is a play conform to σ and thus w is not accepted by automaton A^{det}. Therefore, $q \in P_{\mathrm{rej}}$ and the final test-node belongs to refuter.

$$\begin{tikzcd}
\boldsymbol{\omega} & & \mathbf{f}(\boldsymbol{\omega}) \\
\mathit{Check}(S, P_{\mathrm{rej}}, q_0) \arrow[d] & \rightsquigarrow & S \arrow[d] \\
\vdots \arrow[d] & & \vdots \arrow[d] \\
\mathit{Check}(\varepsilon, P_{\mathrm{rej}}, q) \arrow[d] & \rightsquigarrow & w \\
\mathit{Test}(q, P_{\mathrm{rej}}) & & q_0 \xrightarrow{w} q
\end{tikzcd}$$

Figure 22.: No verify edge in ω.

Second, we consider the case where ω contains at least one verify edge. Refer to Figure 23 for an illustration (left-hand side) and notation. Let $v = \mathit{Verify}(X, P', q)$ be the last verify-node in ω.

We identify prefix $\omega_i \sqsubseteq \omega$ with the prefix ending at C_i for $i = 1, 2, 3$. Then, we have for the (simulated) play prefixes $f(\omega_i)$ that

- $\mathit{last}(f(\omega_1)) = vX\gamma_1\alpha$ with $q_0 \xrightarrow{v} q$ (invariant 1)
- $\mathit{last}(f(\omega_2)) = v\gamma_2\gamma_1\alpha =: v\gamma_2\alpha'$ (definition of f)
- $\mathit{last}(f(\omega_3)) = vw\alpha'$, where $q_0 \xrightarrow{vw} q'$ (invariant 1 + 3)
- The play from $vX\gamma_1\alpha$ to $vw\alpha' = vw\gamma_1\alpha$ is valid and conform to σ (invariant 2).

Consider now prediction P'. By definition of σ', P' contains all states q'' s.t. there exists a valid play conform to σ from $vX\gamma_1\alpha$ to some sentential form of shape $vw'\gamma_1\alpha$. This means that P' in particular contains q'', where $q \xrightarrow{w} q''$ (last bullet above). Then, we have that $q_0 \xrightarrow{v} q \xrightarrow{w} q'$ and because automaton A^{det} is deterministic, it follows that $q' = q'' \in P'$ (third bullet above). Therefore, the test-node belongs to refuter.

$$\begin{tikzcd}
\boldsymbol{\omega} & \mathbf{f}(\boldsymbol{\omega}) \\
\mathit{Check}(S, P_{\text{rej}}, q_0) \arrow[d] \arrow[r, phantom, "\rightsquigarrow"] & S \arrow[d] \\
\vdots \arrow[d] & \vdots \arrow[d] \\
C_1 = \mathit{Check}(X\gamma_1, P, q) \arrow[d] \arrow[r, phantom, "\rightsquigarrow"] & vX\gamma_1\alpha \arrow[ddd] \\
\mathit{Claim}(X\gamma_1, P', P, q) \arrow[d] & \\
v = \mathit{Verify}(X, P', q) \arrow[d] & \\
C_2 = \mathit{Check}(\gamma_2, P', q) \arrow[d] \arrow[r, phantom, "\rightsquigarrow"] & v\gamma_2\alpha' \arrow[d] \\
\vdots \arrow[d] & \vdots \arrow[d] \\
C_3 = \mathit{Check}(\varepsilon, P', q') \arrow[d] \arrow[r, phantom, "\rightsquigarrow"] & vw\alpha' \\
\mathit{Test}(q', P') & q_0 \xrightarrow{vw} q'
\end{tikzcd}$$

Figure 23.: Node v is the last verify-node in ω.

□

We have proven that if refuter wins the inclusion game, she also wins the *GC game*. We show that the converse direction holds as well.

5.5.2. From the *GC game* to the inclusion game.

Let us now assume that we have a positional winning strategy σ' for refuter from $\mathit{Check}(S, P_{\text{rej}}, q_0)$ in $\mathcal{G}_{\text{GC}}$. We derive a winning strategy σ for refuter in $\mathcal{G}$ from σ' by constructing a pushdown transducer $\mathcal{P}$ that reads play prefixes in $\mathcal{G}$ and outputs moves for refuter.

Definition 21
A pushdown transducer is given by the tuple $\mathcal{P} = (\mathcal{Q}_{\mathcal{P}}, \Gamma, \Delta, v_I, \Sigma_I, \Sigma_O, \lambda)$, where

- *$\mathcal{Q}_{\mathcal{P}}$ is the finite set of states,*
- *Γ is the finite stack alphabet, $\Gamma_\perp := \Gamma \uplus \{\perp\}$ for a fresh symbol $\perp \notin \Gamma$,*
- *$\Delta \subseteq \mathcal{Q}_{\mathcal{P}} \times \Gamma_\perp \times \Sigma_I \times \mathcal{Q}_{\mathcal{P}} \times \Gamma_\perp^{\leq 2}$ is the finite set of transitions,*
- *$v_I \in \mathcal{Q}_{\mathcal{P}}$ is the initial state,*
- *Σ_I resp. Σ_O are the finite input resp. output alphabets and*
- *$\lambda : \mathcal{Q}_{\mathcal{P}} \rightharpoonup \Sigma_O$ the partial output function.*

The transitions t can be classified into three categories. Let $t = (v, z, r, v', z')$ where $v, v' \in \mathcal{Q}_{\mathcal{P}}$, $z \in \Gamma_\perp$, $r \in \Sigma_I$ and $z' \in \Gamma_\perp^{\leq 2}$. Then we call t

- *a pop-transition if $|z'| = 0$,*
- *a skip transition if $|z'| = 1$ and*
- *a push transition if $|z'| = 2$.*

We write transitions $t = (v, z, r, v', z')$ as $v \xrightarrow[z/z']{z} v'$.

The symbol $\bot$ is used to mark the bottom of the stack. It can not be pushed on nor popped from the stack.

A configuration of $\mathcal{P}$ is given by an element $c = (v, s)$ with $v \in \mathcal{Q}_\mathcal{P}$ and $s \in \Gamma^+_\bot$. A configuration captures the state and the stack content of $\mathcal{P}$ at a given moment.

A run $\mathcal{C}$ of the pushdown transducer on $R = r_1 \dots r_n \in \Sigma_I^*$ is a sequence of configurations $C = c_1, c_2, \dots$ s.t.

- $c_1 = (v_I, \bot)$
- *if $c_i = (v, xs)$ and $q \xrightarrow[x/y]{r_i} q'$ ($x, y \in \Gamma^{\leq 2}$, $s \in \Gamma^+_\bot$) we have that $c_{i+1} = (v', ys)$. We denote this by $c_i \to c_j$.*

Alternatively, we may write a run as $v_I \xrightarrow[x_1/y_1]{r_1} v_1 \xrightarrow[x_2/y_2]{r_2} v_2 \xrightarrow[x_3/y_3]{r_3} \dots$ where the stack changes are depicted on the arrows. Both representations are equivalent: At the beginning of a run, the stack always consists of only the $\bot$-symbol, thus we can recompose the stack content for any v_i of the run.

First, we define the PDS and then prove that the strategy implemented by $\mathcal{P}$ is indeed a winning strategy for refuter in $\mathcal{G}$.

The pushdown transducer $\mathcal{P}$. Assume that refuter has a winning strategy σ' for $\mathcal{G}_{\text{GC}}$. The idea of the pushdown transducer $\mathcal{P}$ is the following. It should read the moves of prover during the play and output moves for prover. The strategy σ implemented by $\mathcal{P}$ simulates the winning strategy σ' in $\mathcal{G}$. We use the game graph of $\mathcal{G}_{\text{GC}}$ as control structure. More precisely, we only use the parts of the graph that are reachable by plays conform to σ' i.e. if a node belongs to prover, we include all its successors (we need to react to all possible choices of prover) and if it belongs to refuter, we only take the successor given by σ'.

Unfortunately, this control structure only allows us to either enter the verify- or the skip-branch. At a claim-node *Claim*$(X\gamma, P', P, q)$, the verify-branch contains the strategy for the derivation process of X and the skip branches contain the strategy for the rest of the game. For strategy σ, we need the information of both the verify and skip-branches. Therefore, we adapt the transitions of the pushdown transducer and make use of the stack. The transitions of the control structure correspond for most parts to the edges of the game graph. At a claim node *Claim*$(X\gamma, P', P, q)$, we first enter the verify-branch. Therefore, there is a transition to the corresponding verify-node *Verify*(X, P', q) and none into the skip-branches. Additionally, we push the claim-node onto the stack. This allows us to find our way into the correct skip-branch once the verify-branch has been traversed. The verify-branch ends at a test-node of shape *Test*(p, P'), where $p \in P'$ as σ' is winning for refuter. Then, we pop the topmost claim-node of the stack. Using this information, we can restore the first node of the appropriate skip-branch *Check*(γ, P, p) and continue the simulation from there. Figure 24 depicts this idea.

We make this precise in the following definition.

Consider the pushdown transducer $\mathcal{P} = (\mathcal{Q}_\mathcal{P}, \Gamma, \Delta, v_I, \Sigma_I, \Sigma_O, \lambda)$ with the following properties.

- The set of states $\mathcal{Q}_\mathcal{P} = \mathcal{V}\big|_{\sigma'}$ is given by the vertices in $\mathcal{G}_{\text{GC}}$ that can be visited by any play that is conform to σ'.
- The finite stack alphabet $\Gamma = \{$*Claim*$(\gamma, P', P, q) \in \mathcal{Q}_\mathcal{P}\}$ consists of the set of claim-nodes contained in $\mathcal{Q}_\mathcal{P}$.
- We define the transition relation Δ below.
- The starting state corresponds to the starting vertex of the *GC game*, *Check*(S, P_{rej}, q_0).

$$\begin{tikzcd}
 & \vdots & \\
 & \mathit{Check}(X\gamma, P, q) \arrow[d] & \\
 & v_i = \mathit{Claim}(X\gamma, P', P, q) \arrow[dl, "\text{Push } v_i"] \arrow[dr, dotted] & \\
\mathit{Verify}(X, P', q) \arrow[d, "X \rightarrow_G \beta"] & & v_j = \mathit{Check}(\gamma, P, p) \arrow[d] \\
\mathit{Check}(\beta, P', q) \arrow[d] & & \vdots \\
\vdots \arrow[d] & & \\
\mathit{Test}(p, P') \arrow[uuurr, "\text{Pop } v_i"] & &
\end{tikzcd}$$

Figure 24.: Run of a pushdown transducer. Rule $X \rightarrow_G \beta$ is an input if X belongs to prover and an output if X belongs to refuter. The stack operations are depicted along the arrows. At all positions where no input is given, we have ε as input. If no stack operation is depicted, a skip operation is executed pushing the same element that the $\mathcal{P}$ has read back onto the stack.

- The finite input/output alphabet are given by $\Sigma_I = \{X \rightarrow_G \gamma \mid X \in \Box\} \cup \{\varepsilon\}$ resp. $\Sigma_O = \{X \rightarrow_G \gamma \mid X \in \bigcirc\}$.
- The output function $\lambda : \mathcal{Q}_\mathcal{P} \rightharpoonup \Sigma_O$ that determines the next move of refuter.

The transition relation Δ contains the following edges.

- $v = \mathit{Claim}(X\gamma, P', P, q) \xrightarrow[z/vz]{\varepsilon} \mathit{Verify}(\gamma, P', q)$
- $v = \mathit{Verify}(X, P, q) \xrightarrow[z/z]{X \rightarrow_G \gamma} \mathit{Check}(\gamma, P, q)$
  If $X \in \bigcirc$, then $X \rightarrow_G \gamma$ is the output of state v, otherwise it is the input.
- $\mathit{Test}(q, P) \xrightarrow[z/\varepsilon]{\varepsilon} \mathit{Check}(\gamma, P, q)$ if $\bot \neq z = \mathit{Claim}(X\gamma, P', P, q)$
  If $z = \bot$, then there is no outgoing transition from the test state and the run ends.
- otherwise $v \xrightarrow[z/z]{\varepsilon} v'$ if $v, v' \in \mathcal{Q}_\mathcal{P}$, $(v, v') \in \mathcal{V}$ and $\sigma'(v) = v'$ in case v belongs to refuter.

Now it remains to show that the strategy σ implemented by $\mathcal{P}$ is winning for refuter in $\mathcal{G}$. Let $\pi = \alpha_1, \alpha_2, \ldots$ be any play conform to σ in $\mathcal{G}$. For the proof, we need to introduce some notation.

Definition 22
Let $\alpha_i = wX\beta$ and $\alpha_{i+1} = w\gamma\beta$ in π. We define $r_i := X \rightarrow_G \beta$.

- *We call $R = r_1, r_2, \ldots$ the sequence of rules of π. Informally, this is the sequence of grammar rules that the players applied during the play π.*

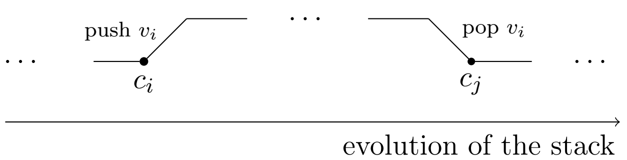


Figure 25.: Evolution of the stack between a pair (c_i, c_j), where $c_i = (v_i, s_i)$ and $c_j = (v_j, s_j)$.

- *We denote by $R_\star$ the subsequence of R restricted to the rules $X \to_G \gamma$ with $X \in \star$ for $\star \in \{\bigcirc, \Box\}$.*
- *Let C be the run of $R_\Box$ on the PDT. Then we call C the run associated with π.*
- *Note that if we take the sequence R' of rules forming the inputs and outputs along the run C, we get $R = R'$. We call R' the sequence of rules of C.*

Let now $C = c_0, c_1 \dots$ be the run associated with π. We prove the claim in two steps.

1. $\pi = \alpha_1, \alpha_2, \dots, \alpha_n$ is finite (and thus also $C = c_0, \dots, c_m$)
2. α_n is winning for refuter i.e. $\alpha_n \in T^*$ and $\alpha_n \notin \mathcal{L}(A)$

Step 1: π is finite We prove the first claim by contradiction. If we assume that π has infinite length, then so has C. From the configuration sequence C, we build a valid play ω in $\mathcal{G}_{\mathrm{GC}}$, which is conform to σ'. We show that ω also has infinite length and obtain a contradiction as σ' is winning for refuter.

To construct the play ω from C, we use the notion of pairs.

Definition 23
Let $c_i = (v_i, s_i)$ and $c_j = (v_j, s_j)$ be two configurations in C. We say that c_i and c_j form a pair if

1. *$v_i = Claim(X\gamma, P', P, p)$ and $v_j = Check(\gamma, P, p')$ with $p' \in P'$ and*
2. *index $j \geq i$ is the smallest index s.t. $s_i = s_j$.*

In other words, after configuration c_i the claim-node v_i is pushed onto the stack and before configuration v_j it is popped again. At state v_i, the run entered the verify-branch, traversing it until a test-node is reached. At v_j, the run enters the corresponding skip-branch. Thus the run between between a pair corresponds to the traversal of the verify branch. Refer to Figure 24 for an example. For all c_k, $k \in \{i+1, \dots, j-1\}$, the stack height s_k is at least as big as $|s_i|$. (see Figure 25) From the definition it is clear that no element of the sequence C can be contained in more than one pair.

Note that the previous statement is about elements of the sequence C, not configurations in general. It is possible that $(c_{i_1}, c_{j_1}), (c_{i_2}, c_{j_2})$ with $(i_1 \neq i_2 \neq j_1 \neq j_2)$ both form a pair and $c_{i_1} = c_{i_2}$ resp. $c_{j_1} = c_{j_2}$.

To construct a valid play π from configuration sequence C, we need to relate the transitions in C to edges in $\mathcal{G}_{\mathrm{GC}}$. The following trivial lemma states the relation.

Lemma 9

1. *If (c_i, c_j) forms a pair, then $v_i = Claim(X\gamma, P', P, q) \to Check(\gamma, P, q') = v_j$ is a valid move conform to σ' in $\mathcal{G}_{GC}$.*
2. *For all transitions $v \xrightarrow[y/z]{} v'$ with $v \neq Test(p, P)$, $v \to v'$ is a valid move conform to σ' in $\mathcal{G}_{GC}$.*

Proof. This follows directly by construction of $\mathcal{P}$. $\square$

We now construct ω from C. Therefore, we inductively construct build $\omega_i \sqsubseteq \omega$ from prefixes $C_i \sqsubseteq C$ of length i.

We maintain the following invariants:

1. ω_i ends v if C_i ends in $c = (v, s)$.

2. ω_i is a valid play in $\mathcal{G}_{\mathrm{GC}}$ conform to σ'.

The idea of the construction is as follows. Suppose that $c_i = (v_i, s_i)$, $c_{i+1} = (v_{i+1}, s_{i+1})$ and that we have already constructed ω_i from C_i. For the most part, transitions in $\mathcal{P}$ correspond to edges in the game graph $\mathcal{G}_{\mathrm{GC}}$. If this is the case, i.e. $(v_i, v_{i+1}) \in \mathcal{E}$, we can simply append v_{i+1} to ω_i. The only exception, where transitions in $\mathcal{P}$ do not correspond to edges in $\mathcal{G}_{\mathrm{GC}}$ is when v_i is a test-node. Consider figure 26.

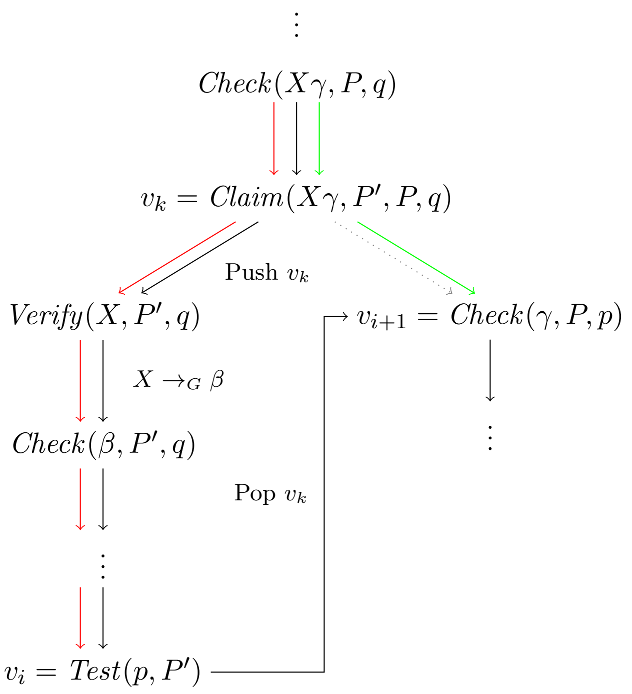


Figure 26.: Deriving play ω_{i+1} from C_{i+1}. The arrows have the following meaning.
(1) The black arrows represent the run C (stack information depicted on the arrows).
(2) The red arrows represent the play prefix ω_i.
(3)The green arrows represent the play prefix ω_{i+1}.

By construction of the $\mathcal{P}$, v_{i+1} is a check-node forming a pair with some claim-node v_k in ω_i. Configuration $c_k = (v_k, s_k)$ is the last configuration in the sequence of some C_k. Between v_k and v_i in ω_i, the run traverses the verify-branch starting at v_k and v_{i+1} is the first node of the corresponding skip-branch starting at v_k. Clearly, (v_k, v_{i+1}) is a valid move in $\mathcal{G}_{\mathrm{GC}}$ and conform to σ'. Instead of prolonging ω_i, we prolong ω_k by v_{i+1} to construct ω_{i+1} i.e. we avoid the whole verify-branch and go directly into the skip-branch.

We now make this precise.

Base case: $i = 1$

Then we have $v_1 = Check(S, P_{\mathrm{rej}}, q_0)$ and define $\omega_1 = v_1$, which satisfies both invariants.

Induction step: $i \to i + 1$

We have to consider two cases:

- Case 1: $|s_{i+1}| = |s_i| - 1$ (top element of the stack is popped).

  This means that $v_{i+1} = \mathit{Check}(\gamma, P, q)$ and v_{i+1} forms a pair with some $v_j = \mathit{Claim}(X\gamma, P', P, q')$ $(j < i)$. We define $\omega_{i+1} = \omega_j, v_{i+1}$, which satisfies the first invariant. By the induction hypothesis and Lemma 9, ω_{i+1} is a valid play in $\mathcal{G}_{\mathrm{GC}}$ conform to σ'.

- Case 2: otherwise.

  In this case, we simply append the state v_{i+1} of v_{i+1} to ω_i and get $\omega_{i+1} = \mathit{playW}_i, v_{i+1}$. The first invariant is satisfied. As $|s_{i+1} \neq |s_i| - 1|$, no pop operation has been executed during the transition from c_i to c_{i+1}. Therefore, v_i cannot be a test node and we can apply 9 which ensures that the second invariant holds.

We now prove that the resulting play ω has infinite length. This leads to the desired contradiction with the fact that σ' is a winning strategy for refuter in $\mathcal{G}_{\mathrm{GC}}$.

Lemma 10
Play ω constructed in the previous step is of infinite length.

Proof. We prove by induction over $l \in \mathbb{N}$, that there exists a k_l s.t. $|\omega_k| \geq l$ for all $k \geq k_l$.
Base case: $l = 1$
For all ω_i, $i \in \mathbb{N}$, it trivially holds that $\omega_i \geq 1$. Thus we fix $k_1 = 1$.
Induction step: $l \to l + 1$
From the induction hypothesis, we know that $|\omega_k| \geq l$ for all $k \geq k_l$. Let $\mathit{last}(C_k) = c_k = (v_k, s_k)$. We consider the induction step where ω_k for $k > k_l$ is constructed by appending v_k to some ω_j for $j < k$.
There are two possible cases.

- ω_k has been constructed by appending v_k to ω_j for $j \geq k_l$. From the induction hypothesis, we know that $|\omega_j| \geq l$ and thus $|\omega_k| > l$.

- ω_k has been constructed by appending v_k to ω_j for $j < k_l$. In this case, we can only deduce that $|\omega_k| \geq l$ using the induction hypothesis. But this can only happen for finitely many $k \geq k_l$ as there are only finitely many $j < k_l$ and each claim-node can only be contained in at most one pair.

Thus, there exists some $k_{l+1} \geq k_l$ s.t. $|\omega_k| \geq l + 1$ for all $k \geq k_{l+1}$.

□

Step 2: α_n is winning for refuter Finally, we prove the second claim that $\alpha_n \in T^*$ and $\alpha_n \notin \mathcal{L}(A)$. Before tackling the claim, we state a few properties of the run C which are necessary for the following proofs. The properties follow from the construction of $\mathcal{P}$ from the *GC game*, but we omit the technical proof here. The enthusiastic reader may convince himself by conducting a formal proof.

Observation. Let (c_i, c_j) be a pair in C, where $v_i = \mathit{Claim}(X\alpha, P, P', q)$ and $v_j = \mathit{Check}(\alpha, P', q')$. Then the successor of v_i is a verify-node $v_{i+1} = \mathit{Verify}(X, P', q)$. Furthermore, between c_i and c_{i+1}, node v_i is pushed onto the stack.
Let now $X \to_G \gamma$ be the grammar rule that is read (if X belongs to prover) or outputted (if X belongs to refuter) after v_{i+1}. We denote by $l = |\gamma|$ the size of the right-hand side of the rule.
We use the structure given by the *GC game* to argue that the sequence $c_i, \ldots, c_j$ in C contains configurations $c_{j_1}, \ldots, c_{j_{l+1}}$ s.t. $v_{j_k} = \mathit{Check}(\gamma_{[k,l]}, P, q_k)$ and $s_{j_k} = s_{i+1}$. By $\gamma_{[k,l]}$, we denote the suffix $\gamma_k, \ldots, \gamma_l$ of γ. In other words, we have such a check-node for each suffix of γ in the sequence $c_i, \ldots, c_j$. In the subsequent proofs, we will reason about these check-nodes contained between the configurations of a pair.
We use an inductive argument to explain why such configurations $c_{j_1}, \ldots, c_{j_{l+1}}$ exist.

Configuration $c_{j_1} = c_{i+2}$ is the direct successor of c_{i+1} in C (see Figure 27). Transition $v_{i+1} \xrightarrow[z/z]{X \to_G \gamma} v_{i+2} = \mathit{Check}(\gamma, P, q)$ corresponds directly to the edge $v_{i+1} \longrightarrow v_{i+2}$ in the *GC game* (see Definition 21). As the stack is not altered in the transition, $s_{i+1} = s_{j_1}$, the claim follows.

$$\begin{tikzcd}
v_i = \mathit{Claim}(X\alpha, P, P', q) \arrow[d, "\text{Push } v_i"] \\
v_{i+1} = \mathit{Verify}(X, P, q) \arrow[d, "X \Rightarrow \gamma"] \\
v_{j_1} = \mathit{Check}(\gamma, P, q)
\end{tikzcd}$$

Figure 27.

Suppose now that we have proven the existence of configuration c_{j_k} in the sequence $c_i, \dots, c_j$. We want to conclude that this is also the case for $c_{j_{k+1}}$. We have to make a case distinction based on $v_{j_k} = \mathit{Check}(\gamma_{[k,l]}, P, q_k)$. The first symbol of $\gamma_{[k,l]}$ is either a terminal or a non-terminal, i.e. $\gamma_{[k,l]} = a\gamma_{[k+1,l]}$ or $\gamma_{[k,l]} = Y\gamma_{[k+1,l]}$.

In the first case, the direct successor of c_{j_k} is the configuration that we look for. The state v_{j_k+1} is of shape $\mathit{Check}(\gamma_{[k+1,l]}, P, q_{k+1})$ (see Figure 28). As in the base case, the transition $v_{j_k} \xrightarrow[z/z]{\varepsilon} v_{j_k+1}$ corresponds directly to an edge in *GC game* by definition of the pushdown transducer.

$$\begin{tikzcd}
v_{j_k} = \mathit{Check}(\gamma_{[k,l]}, P, q_k) \arrow[d] \\
v_{j_k+1} = \mathit{Check}(\gamma_{[k+1,l]}, P, q_{k+1})
\end{tikzcd}$$

Figure 28.

The second case is more involved.

We make use of the relation between $\mathcal{P}$ and the *GC game* again. Consider therefore Figure 29. It depicts the a snippet of the *GC game* starting at vertex v_{j_k} (if we ignore the input/output, the stack operations and the transition from the test-node to $v_{j_{k+1}}$). At the claim-node v_{j_k+1}, the pushdown transducer enters the verify-branch while pushing v_{j_k+1} onto the stack. In the verify-branch, further stack operations may be conducted, but $v_{j_{k+1}}$ can only popped from the stack by the transition from the test-node $\mathit{Test}(p_k, P_k)$ to $v_{j_{k+1}}$. In this case, the run reaches state $v_{j_{k+1}}$ and the stack is of form $s_{j_{k+1}} = s_{j_k}$. Thus, $c_{j_{k+1}} = (v_{j_{k+1}}, s_{j_{k+1}})$ is the desired configuration.

$$\begin{tikzcd}
 & \vdots & \\
 & v_{j_k} = \mathit{Check}(Y\gamma_{[k+1,l]}, P, q_k) \arrow[d] & \\
 & v_{j_k+1} = \mathit{Claim}(Y\gamma, P_k, P, q_k) \arrow[dl, "\text{Push } v_{j_{k+1}}"] \arrow[dr, dotted] & \\
\mathit{Verify}(Y, P_k, q_k) \arrow[d, "X \to_G \beta"] & & v_{j_{k+1}} = \mathit{Check}(\gamma_{[k+1,l]}, P, p_k) \arrow[d] \\
\mathit{Check}(\beta, P_k, q_k) \arrow[d] & & \vdots \\
\vdots \arrow[d] & & \\
\mathit{Test}(p_k, P_k) \arrow[uuurr, "\text{Pop } v_{j_{k+1}}"'] & &
\end{tikzcd}$$

Figure 29.

Thus, we only have to argue that the run does not get stuck in the verify-branch, i.e. $v_{j_{k+1}}$ will be popped at some configuration in the sequence $c_i, \dots, c_j$. This is guaranteed by the fact that $|s_j| = |s_{i+1}| - 1 = |s_{j_k}| - 1$.

We can argue in a similar fashion to get a weaker claim in case we do not want to assume that C contains pair (c_i, c_j), but only a part of the sequence $c_i, \dots, c_j$. Suppose therefore that $c_i = (v_i, s_i)$ with $v_i = \mathit{Claim}(X\alpha, P, P', q)$. Let $X \to_G \gamma$ be the grammar rule which is inputted/outputted after c_{i+1}. Let then $c_{j_{n+1}} = (v_{j_{n+1}}, s_{j_{n+1}})$ s.t. $v_{j_{n+1}} = \mathit{Check}(\gamma_{[n+1,l]}, P, q_{n+1})$ and $s_{j_{n+1}} = s_{i+1}$. Then, we can conclude in a similar fashion as above that the sequence $c_i, \dots, c_{j_{n+1}}$ contains configurations $c_{j_1}, \dots, c_{j_n}$ s.t. $v_{j_k} = \mathit{Check}(\gamma_{[k,l]}, P, q_k)$ and $s_k = s_{j_{n+1}}$ for $k = 1, \dots, n$.

We now use the observations to prove the first lemma.

Lemma 11
Let $c_i = (v_i, s_i)$ and $c_j = (v_j, s_j)$ be two configurations in C with $v_i = \mathit{Claim}(X\gamma, P', P, q)$ and $v_j = \mathit{Check}(\gamma, P, q')$, $q' \in P'$. Let $R = r_1, \dots, r_n$ be the restriction the sequence of rules of C to the inputs/outputs between c_i and c_j in C. If (c_i, c_j) forms a pair then $X \Rightarrow^* v$, $v \in T^*$ with $q \xrightarrow{v} q'$ by applying sequence R starting from X.

Proof. We prove this by induction over the maximal stack growth g in the sequence $c_i, \dots, c_j$. In other words, if the maximal stack size of any configuration $c_k = (v_k, s_k)$ in the sequence $c_i, \dots, c_j$ is $|s_k| = |s_i| + g$.

Base case: $g = 1$

As v_i is pushed onto the stack after c_i and only popped before c_j, no additional push can be performed during the sequence $c_i, \dots, c_j$. Figure 30 depicts this situation. This entails that the grammar rule $X \Rightarrow \beta$ which is applied after the verify-node v_{i+1} is of a special form. The derived sentential form must be a terminal word, i.e. $\beta = w = w_1 \dots w_l \in T^*$. If this is not the case, meaning that β contains a non-terminal, we would need an additional claim-node in the sequence $v_i, \dots, v_j$, which would bring on an additional push-operation. After the verify-node v_{i+1}, the derived word w is consumed symbol by symbol in the consecutive check-nodes and the state changes are composed. If $v_k = \mathit{Check}(w_{[k,l]}, P, q_k)$, then $v_{k+1} = \mathit{Check}(w_{[k+1,l]}, P, q_{k+1})$ with $q_k \xrightarrow{w_k} q_{k+1}$. Here, we use notation $w_{[k,l]}$ for $w_k \dots w_l$. By an inductive argument, we get that the last check-node $\mathit{Check}(\varepsilon, P, q_{l+1})$ before the test-node contains state q_l s.t. $q \xrightarrow{w} q_{l+1}$. As $X \Rightarrow v$, $v \in T^*$ is the only rule in sequence $c_i, \dots, c_j$, the claim follows.

$$\begin{tikzcd}
v_i = \mathit{Claim}(X\alpha, P, P_0, q_1) \arrow[d, "\text{Push } v_i"] \\
v_{i+1} = \mathit{Verify}(X, P, q_1) \arrow[d, "X \Rightarrow w"] \\
\mathit{Check}(w, P, q_1) \arrow[d] \\
\mathit{Check}(w_{[2,l]}, P, q_2) \arrow[d] \\
\vdots \arrow[d] \\
\mathit{Check}(\varepsilon, P, q_{l+1}) \arrow[d] \\
\mathit{Test}(q_{l+1}, P) \arrow[d, "\text{Pop } v_i"] \\
v_j = \mathit{Check}(\gamma, P_0, q_{l+1})
\end{tikzcd}$$

Figure 30.: Sequence $v_i, \dots, v_j$.

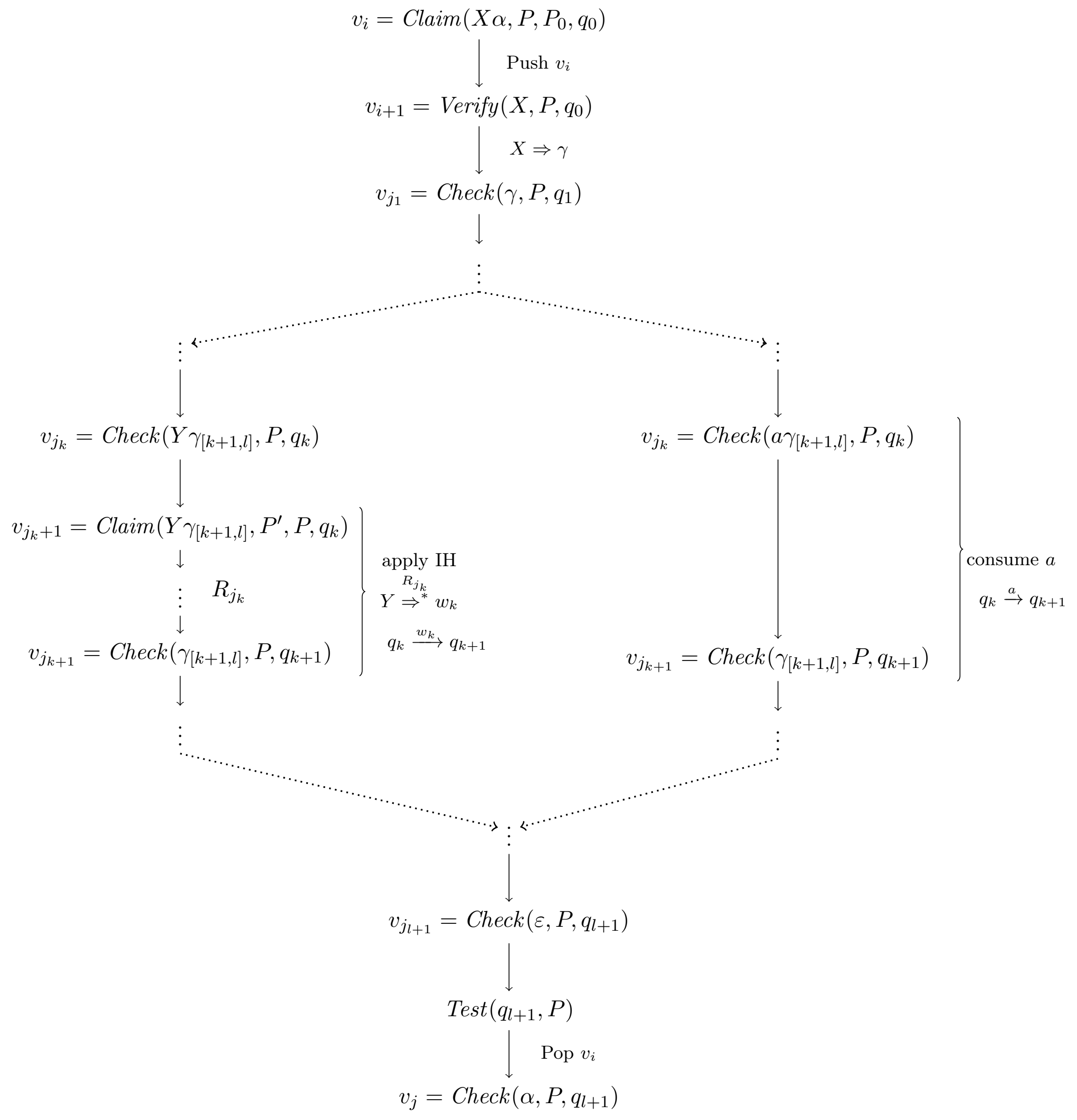
Induction step: $g \to g + 1$

Let $X \to_G \beta$ be the rule applied after the verify-node $v_{i+1} = \mathit{Verify}(XP', q)$. In this case, β may contain non-terminals and thus sequence $v_{i+1}, \dots, v_j$ includes claim-nodes.

We first introduce some notation and gather some properties of the sequence $c_i, \dots, c_j$. (Figure 31). All claims follow by construction of $\mathcal{P}$ from the *GC game* (see Observation above).

- Let $l = |\beta|$ be the size of the right-hand side of grammar rule $X \to_G \beta$ which is inputed/outputed after configuration c_{i+1}.
- Let h be the stack height after v_i has been pushed, i.e. $h = |s_{i+1}| = |s_i| + 1$.
- Let $I = \{j_1, \dots, j_{l+1}\} \subseteq \{i, \dots, j\}$ be the set of indices s.t. for configuration $c_{j_k} = (v_{j_k}, s_{j_k})$, $k = 1, \dots, n$ holds that v_{j_k} is a check-node and $|s_{j_k}| = h$. Note that $|I| = |\gamma| + 1$ as we have one such check-node for each suffix of γ. More precisely, the check nodes v_{j_k} are of shape $\mathit{Check}(\gamma_{[k,l]}, P, q_k)$.
- Let $v_{j_k} = \mathit{Check}(a\gamma_{[k+1,l]}, P, q_k)$, i.e. the first symbol of $\gamma_{[k,l]}$ is a terminal. Then, the terminal a is consumed and the successor of v_{j_k} is $v_{j_{k+1}} = \mathit{Check}(a\gamma_{[k+1,l]}, P, q_{k+1})$ where $q_k \xrightarrow{a} q_{k+1}$. This situation is depicted on the right in Figure 31. We collect the indices j_k of these check-nodes in $I_T \subseteq I$.
- Let now the first symbol of $\gamma_{[k,l]}$ be a non-terminal X, i.e. $v_{j_k} = \mathit{Check}(Y\gamma_{[k+1,l]}, P, q_k)$. Then, its successor $v_{j_k+1} = \mathit{Claim}(Y\gamma_{[k+1,l]}, P', P, q_k)$ forms a pair with $v_{j_{k+1}} =$

Check$(\gamma_{[k+1,l]}, P, q_{k+1})$. We call R_{j_k} the sequence of rules contained in the subsequences $v_{j_k+1}, \ldots, v_{j_{k+1}}$. Furthermore we denote by w_k the sentential form derived from Y by the sequence R_{j_k}. This situation is depicted on the left in Figure 31. We collect the indices j_k of these check-nodes in $I_N \subseteq I$. Note that we have $I_T \uplus I_N = I$.



$v_i = Claim(X\alpha, P, P_0, q_0)$
Push v_i
$v_{i+1} = Verify(X, P, q_0)$
$X \Rightarrow \gamma$
$v_{j_1} = Check(\gamma, P, q_1)$
$v_{j_k} = Check(Y\gamma_{[k+1,l]}, P, q_k)$
$v_{j_k+1} = Claim(Y\gamma_{[k+1,l]}, P', P, q_k)$
R_{j_k}
$v_{j_{k+1}} = Check(\gamma_{[k+1,l]}, P, q_{k+1})$
apply IH
$Y \overset{R_{j_k}}{\Rightarrow^*} w_k$
$q_k \xrightarrow{w_k} q_{k+1}$
$v_{j_k} = Check(a\gamma_{[k+1,l]}, P, q_k)$
$v_{j_{k+1}} = Check(\gamma_{[k+1,l]}, P, q_{k+1})$
consume a
$q_k \xrightarrow{a} q_{k+1}$
$v_{j_{l+1}} = Check(\varepsilon, P, q_{l+1})$
$Test(q_{l+1}, P)$
Pop v_i
$v_j = Check(\alpha, P, q_{l+1})$

Figure 31.: States $v_{j_k}, v_{j_{k+1}}$ $(j_k, j_{k+1} \in I)$ in the sequence $v_i, \ldots, v_j$. The left part represents the case where $v_{j_k} = Check(Y\gamma_{[k+1,l]}, P, q_k)$ and the right part where $v_{j_k} = Check(a\gamma_{[k+1,l]}, P, q_k)$.

For all pairs $(v_{j_k+1}, v_{j_{k+1}})$, $(j_k \in I_N)$ in the sequence $v_i, \ldots, v_j$, we can apply the induction hypothesis as their maximal stack growth is smaller by at least 1 (we pushed at the beginning of sequence $v_i, \ldots, v_j$). Let $v_{j_k+1} = Claim(Y\gamma_{[k+1,l]}, P', P, q_k)$ and $v_{j_{k+1}} = Check(\gamma_{[k+1,l]}, P, q_{k+1})$. Then, we get that w_k $(Y \Rightarrow^* w_k)$ is a terminal word s.t. $q_k \xrightarrow{w} q_{k+1}$.

We now put the collected facts together to show that $X \Rightarrow^* w$ by applying sequence of rules R and that $w \in T^*$.

Therefore, we show the following lemma by induction over $k = 1, \ldots, l+1$.

Lemma 12

Let $v_k = Check(\gamma_{[k,l]}, P, q_k)$. Then we have that $X \Rightarrow^* v_k.\gamma_{[k,l]}$ for some terminal word $u_k \in T^*$ by the sequence of rules $X \rightarrow_G \gamma, R_{t_1}, \ldots, R_{t_n}$ where $t_1, \ldots, t_n \in I_N$, $t_1, \ldots, t_n \leq j_k$. For v_k furthermore holds that $q_0 \xrightarrow{u_k} q_k$.

Proof. **Base case:** $k = 1$

We know that $q_{j_1} = \mathit{Check}(\gamma_{[1,l]}, P, q_0) = \mathit{Check}(\gamma, P, q_0)$. Because q_{j_1} is the first check node after the verify-node v_{i+1}, only rule $X \to_G \gamma$ has been applied so far. The claim follows as $\varepsilon.\gamma = \gamma$ and $q_0 \xrightarrow{\varepsilon} q_0$.

Induction step: $k \to k + 1$

We have to distinguish two cases:

1. $v_{j_{k+1}}$ is part of a pair, i.e. $v_{j_k} \in I_N$. Then we know that

$$\begin{aligned} v_{j_k} &= \mathit{Check}(Y\gamma_{[k+1,l]}, P, q_k) \\ v_{j_{k+1}} &= \mathit{Check}(\gamma_{[k+1,l]}, P, q_{k+1}) \text{ with } q_k \xrightarrow{w_k} q_{k+1} \end{aligned}$$

   Recall that $w_k \in T^*$ is derived from Y by the sequence of rules R_{j_k} (Induction hypothesis of Lemma 11).

   From the induction hypothesis of Lemma 12, we get that

$$X \Rightarrow^* u_k.\gamma_{[k,l]} \text{ where } q_0 \xrightarrow{u_k} q_k$$

   by $X \Rightarrow \gamma, R_{t_1}, \dots, R_{t_n}$ $(t_1, \dots, t_n \in I_N,\ t_1, \dots, t_n \leq j_k)$. Furthermore, we just argued that $\gamma_k = Y \Rightarrow^* w_k$ by rules R_k. We can easily deduce that

$$\begin{aligned} X &\Rightarrow^* u_k.w_k.\gamma_{[k+1,l]} =: u_{k+1}.\gamma_{[k+1,l]} \\ &\text{where } q_0 \xrightarrow{u_k} q_k \xrightarrow{w_k} q_{k+1} \end{aligned}$$

   by $X \Rightarrow \gamma, R_{t_1}, \dots, R_{t_n}, R_{j_k}$.

2. $v_{j_{k+1}}$ is not part of a pair. Then we know that

$$\begin{aligned} v_{j_k} &= \mathit{Check}(a\gamma_{[k+1,l]}, P, q_k) \\ v_{j_{k+1}} &= \mathit{Check}(\gamma_{[k+1,l]}, P, q_{k+1}) \text{ with } q_k \xrightarrow{a} q_{k+1} \end{aligned}$$

   From the induction hypothesis, we get that

$$X \Rightarrow^* u_k.\gamma_{[k,l]} \text{ where } q_0 \xrightarrow{u_k} q_k$$

   by $X \Rightarrow \gamma, R_{t_1}, \dots, R_{t_n}$ $(t_1, \dots, t_n \in I_N,\ t_1, \dots, t_n \leq j_k)$.

   By construction, there can be no rules between v_{j_k} and $v_{j_{k+1}}$. It follows directly that

$$\begin{aligned} X &\Rightarrow^* u_k.\gamma_k.\gamma_{[k+1,l]} = u_k.a.\gamma_{[k+1,l]} := u_{k+1}.\gamma_{[k+1,l]} \\ &\text{where } q_0 \xrightarrow{u_k} q_k \xrightarrow{a} q_{k+1} \end{aligned}$$

   by $X \Rightarrow \gamma, R_{t_1}, \dots, R_{t_n}$.

□

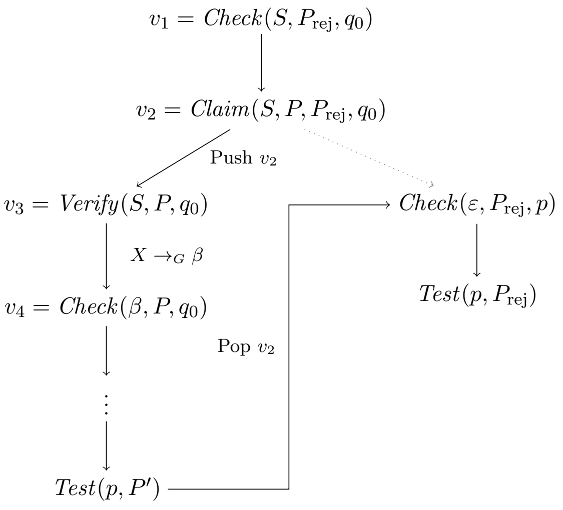
Using this for $k = l + 1$, where $v_{k+1} = \mathit{Check}(\varepsilon, P, q_{l+1})$ induces that $X \Rightarrow^* u_{l+1}.\varepsilon = u_{l+1}$ with $q_0 \xrightarrow{u_{l+1}} q_{l+1}$ by sequence of rules R. The claim follows. □

However this does not suffice to show that α_n is winning for refuter. We want to apply Lemma 11 to the outermost pair (c_i, c_j) contained in C. In case $c_i = (\mathit{Claim}(S, P, P_{\text{rej}}, \rho_\varepsilon), \bot)$ and $c_j = (\mathit{Check}(\varepsilon, P_{\text{rej}}, \rho_v), \bot)$, the claim follows. But, we cannot simply assume that this is the case. The fact that we constructed $\mathcal{P}$ from a winning strategy for the *GC game* guarantees us that we do not get stuck in a verify-branch. However, the run C may get stuck at a verify-node if the grammar rules that have been read/outputted so far already derived a terminal word w from S and C never reaches configuration c_j. The following lemma shows that this can not be the case.

Lemma 13
Let $c = (v, s)$ be a configuration in C where $v = \mathit{Verify}(X, P, q)$. Let furthermore R be the sequence of rules restricted to the partial run $c_1, \ldots, c$. Then, $S \Rightarrow_G wX\alpha$ by applying sequence R to S. We say that sentential form $wX\alpha$ is associated to c.

Proof. We prove this by induction over the stack height $h = |S|$ of configuration c.
Base case: $h = 1$

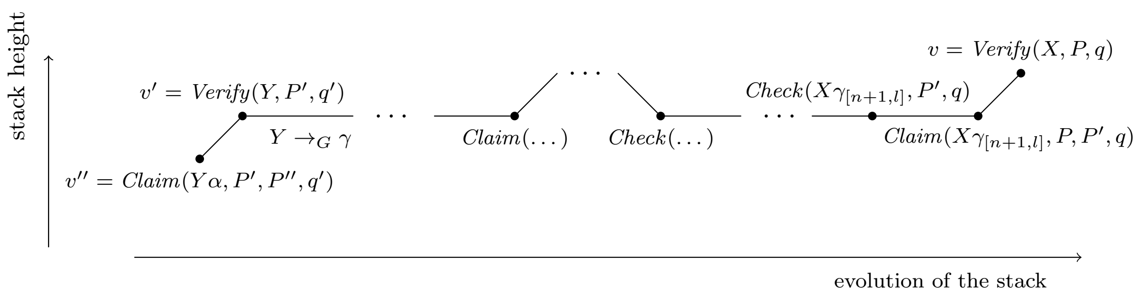
The only configuration c s.t. v is a verify node and $|s| = 1$ is c_3 where $v_3 = \mathit{Verify}(S, P, P_{\text{rej}}, q_0)$. Consider Figure 32 for a graphic representation. After configuration c_2, v_2 is pushed onto the stack. A verify-node can only occur as direct successor of a claim-node and during this transition a push-operation is performed. Thus, no configuration with a verify-node as state and stack height 1 can appear as long as v_2 is already on the stack. But in case v_2 gets popped, no further configurations with verify-nodes as states appear (see Figure 32). Therefore, $c = c_3$ is the only configuration with the desired properties.



$v_1 = \mathit{Check}(S, P_{\text{rej}}, q_0)$
$v_2 = \mathit{Claim}(S, P, P_{\text{rej}}, q_0)$
Push v_2
$v_3 = \mathit{Verify}(S, P, q_0)$
$X \to_G \beta$
$v_4 = \mathit{Check}(\beta, P, q_0)$
Pop v_2
$\mathit{Test}(p, P')$
$\mathit{Check}(\varepsilon, P_{\text{rej}}, p)$
$\mathit{Test}(p, P_{\text{rej}})$

Figure 32.

As no rules are read/outputted in the sequence c_1, c_2, c_3, $R = \varepsilon$ and the sentential form associated with c is S. Thus, the claim holds.
Induction step: $h \to h + 1$
Let $c = (v, s)$ be s.t. $v = \mathit{Verify}(X, P, q)$ and $|s| = h + 1$. Let now $c' = (v', s')$ be the configuration with a verify-node as state, $v' = \mathit{Verify}(Y, P', q')$, and $|s'| = h$ s.t. no other configuration with the same properties is located between c' and c. Let finally c'' be the direct predecessor of c'. Thus $v'' = \mathit{Claim}(Y\alpha, P', P'', q')$ and $|s''| = h - 1$.
We now state a few properties of the sequence $c'', \ldots, c$. Refer to Figure 33 for a graphical representation of the sequence.



stack height
$v' = \mathit{Verify}(Y, P', q')$
$Y \to_G \gamma$
$v'' = \mathit{Claim}(Y\alpha, P', P'', q')$
$\mathit{Claim}(\ldots)$
$\mathit{Check}(\ldots)$
$\mathit{Check}(X\gamma_{[n+1,l]}, P', q)$
$v = \mathit{Verify}(X, P, q)$
$\mathit{Claim}(X\gamma_{[n+1,l]}, P, P', q)$
evolution of the stack

Figure 33.: Sequence $c'', \ldots, c$.

- Let $Y \to_G \gamma$ be the grammar rule that is read/inputted after c'. We denote by l the length of the right-hand side of the rule.
- The sequence $c'', \ldots, c$ may contain several pairs (c_i, c_j) with $|s_i| = |s_j| = h$. Thus, we can apply Lemma 11 to all such pairs in $c'', \ldots, c$.

- The two direct predecessors of c have states $\mathit{Claim}(X\gamma_{[n+1,l]}, P, P', q)$ resp. $\mathit{Check}(X\gamma_{[n+1,l]}, P', q)$ for some $k \leq l$ and stack height h. Let c_{j_k} be the configuration corresponding to the check-node.
- Using the observation we made at the beginning of this paragraph, we get that the sequence $c'', \ldots, c$ contains configurations $c_{j_1}, \ldots, c_{j_{n-1}}$ s.t. s.t. $v_{j_k} = \mathit{Check}(\gamma_{[k,l]}, P, q_k)$ and $s_k = s_{j_{n+1}}$ for $1 \leq k \leq n-1$.
- Let R_2 be the sequence of rules restricted to the sequence $c', \ldots, c$.

The stated properties induce that we have a similar setting than the one in Lemma 11. The only difference is that we do not want to assume that the configurations $c_{j_1}, \ldots, c_{j_n}$ are embedded in a pair. However, we do not need this assumption to apply Lemma 12 to $\mathit{Check}(X\gamma_{[n+1,l]}, P', q)$. We get that $Y \Rightarrow^* w_2 X\gamma_{[n+1,l]}$ by applying rules R_2 to Y for some $w_2 \in T^*$.

It remains to combine this statement with the induction hypothesis applied to c'. Let therefore R_1 be the sequence of rules restricted to $c_1, \ldots, c'$. Then we can deduce by the induction hypothesis that the sentential form associated with c' is of shape $w_1 Y\beta$ for some $w_1 \in T^*$. It follows that the sequence of rules $R = R_1, R_2$ applied to S entails derivation $S \Rightarrow^* w_1 Y\beta \Rightarrow^* w_1 w_2 X\gamma_{[n+1,l]}\beta$ which is of the desired shape.

□

It remains to show that α_n is winning for refuter.

Lemma 14
$\alpha_n \in T^*$ and $\alpha_n \notin \mathcal{L}(A^{det}) = \mathcal{L}(A)$.

Proof. We apply the Lemma 11 to the outermost pair contained in $C = c_0 \ldots, c_m$.

By Lemma 13, we get that the pair consists of $c_1 = (\mathit{Claim}(S, P, P_{\text{rej}}, \rho_\varepsilon), \bot)$ and $c_{m-1} = (\mathit{Check}(\varepsilon, P_{\text{rej}}, \rho_v), \bot)$. As there are no rules between $c_0 = (\mathit{Check}(S, P_{\text{rej}}, \rho_\varepsilon), \bot)$ and c_1 as well as between c_{m-1} and $c_m = (\mathit{Test}(\rho_v, P_{\text{rej}}), \bot)$, all the rules R in C are contained between the pair. Applying Lemma 11 gives us that $S \Rightarrow^* v = \alpha_n$ with $\alpha_n \in T^*$ by rules R. As all runs in $\mathcal{P}$ are also winning plays of $\mathcal{G}_{\text{GC}}$ by construction, $\rho_{\alpha_n} \in P_{\text{rej}}$ and thus $\alpha_n \notin \mathcal{L}(A^{\text{det}})$.

□

5.6. Overview of the Guess & Check approach

We end the section again by presenting a brief overview of the algorithm to solve context-free games. Let $\mathscr{G}$ be a context-free game given by the context-free grammar G and the finite automaton A. We employ the following steps.

1. Determinize the finite automaton A into A^{det}.
2. Construct the finite game graph $\mathcal{G}_{\text{GC}}$ from the grammar G and the determinized automaton A^{det}.
3. Determine the winner of the *GC game* by an attractor $\mathsf{Attr}_{\mathbb{A},\bigcirc}(\mathcal{V}_{\text{F}}, \mathcal{E})$,where $\mathcal{V}_{\text{F}}$ contains all test-nodes $\mathit{Test}(q, P)$ that are winning for refuter, i.e. $q \in P$ and $\mathcal{E}$ are the edges of $\mathcal{G}_{\text{GC}}$. If the starting state $\mathit{Check}(S, P_{\text{rej}}, q_0)$ is contained in the attractor, refuter wins. Otherwise, prover wins.

Part II.

Comparison

6. Overview of the comparison

In the first part of the thesis, we defined three methods to solve context-free games. It is an interesting question whether these three approaches, while solving the same problem, also have common design principles or whether and how they differ. We focus on three aspects.

1. **Computation of possible plays vs. winning plays.**
   When dealing with context-free games, there are two informations which need to be computed. First, we need to know which plays are possible in general and how the players influence them, without regard of whether they are winning for refuter or not. Second, we have to determine whether refuter can enforce one of the winning plays. We will see how the three approaches deal with this two different tasks and whether they handle them separately or at the same time.

2. **Dealing with non-determinism.**
   In general, we can not assume that the automaton A on the right-hand side is deterministic. Thus, we examine how the algorithms deal with a non-deterministic automaton.

3. **The computed (intermediary) information** The last question that we address is whether similar information is computed during the algorithms. All three algorithms work with some kind of fixed-point iteration. The summary approach uses a Kleene iteration to compute the formulas, the saturation approach saturates an automaton and the Guess & Check approach computes an attractor. We compare both the information of intermediary and the final step of the three approaches. This gives us an insight into the differences and similarities of the mechanics of the algorithms.

   We use the summary method as central point and compare it the two other approaches. More precisely, we determine whether and how the formulas of the summary approach can be found in the information computed by the other methods. Furthermore, we determine whether any additional information (compared to the formulas) is computed by the Saturation resp. Guess & Check approach. For each of the results, we discuss how it can be explained by the mechanics of the respective algorithms.

In the following sections, we successively address the aspects above.

7. Computation of possible plays vs. winning plays

In this section, we examine whether the algorithms handle the tasks of computing the possible plays and determining the winning plays separately.

Summary approach In the summary approach, the separate computation of the possible plays in form of a formula and the actual winner by evaluating the formula was a key aspect to the method. This separation allowed us to examine the parse trees rather than the game arena. The parse tree does not allow us to determine the winner of the play directly (for example by an attractor) as the leafs are not labeled the derivable terminal prefixes. The parse trees have as leafs the terminals $w_1, \ldots, w_n$ that constitute the derivable words w. In the Kleene iteration, we assemble the boxes of the derivable words from the boxes of the terminals. Thus, the first part of the computation is mostly conducted on the grammar, more precisely on the rules of the grammar. The automaton A is only used to factorize the domain to make it finite. After we determined the boxes of all terminals $a \in T$, the automaton is not needed again.

Saturation approach The saturation approach works in two steps as well. In the first step, we saturate the determinized automaton A^{det} by adding new transitions labeled by non-terminals to the automaton. The automaton is again used to factorize the domain. The goal is that after the factorization, A^{det} accepts all sentential forms contained in the attractor $\mathsf{Attr}_{\bigcirc}(\overline{\mathcal{L}(A^{\text{det}})}, \rightarrow_{\mathsf{B}_G})$, where $\rightarrow_{\mathsf{B}_G}$ are the edges of the game arena. The saturation is based solely on the grammar rules and the transitions of A^{det} without regard for final or non-final states. Thus, we have computed runs for all sentential forms contained in the game arena in the saturated automaton, regardless of whether they are contained in the attractor or not. Only in the second step, where we determine whether the starting symbol S has an accepting run, we compute which player wins the game.

Guess & Check approach The Guess & Check algorithm however does not use this separation. This becomes clear at the first claim-nodes in the game graph $\mathcal{G}_{\text{GC}}$. These nodes are of shape $\mathit{Claim}(S, P', P_{\text{rej}}, q_0)$, where S is the starting symbol of the grammar, $P' \subseteq 2^Q$ is any prediction, P_{rej} is the prediction containing all non-final states and q_0 the initial state of automaton A^{det}.

In the verify-branch of the claim-node, it is checked whether refuter can enforce that every play from S ends in a terminal word w s.t. $q_0 \xrightarrow{w} q$ with $q \in P'$. To this end, the derivation process of S needs to be played out and thus the verify-branch contains all possible plays. In the skip-branches, we check whether the prediction P' is also favorable for refuter. For every $p \in P'$, we have a test-node $\mathit{Test}(p, P_{\text{rej}})$ which belongs to refuter if p is not a final state. Together, these branches decide whether refuter wins the game. By the attractor, we simultaneously determine whether prediction P' is valid (verify-branch), which concerns the possible plays, and whether refuter wins with this prediction (skip-branch), which determines if the plays are winning. As the claim-nodes belong to prover, $\mathit{Claim}(S, P', P_{\text{rej}}, q_0)$ is only contained in the attractor if all successor nodes are i.e. refuter wins in all branches.

8. Non-determinism

In this section, we examine how the three approaches deal with a non-deterministic automaton on the right-hand side of the inclusion. We will see that actually all of the methods use some kind of determinization.

Summary approach Although the summary approach can not completely avoid determinizing computations, it is able to avoid an upfront determinization of A in favor of an on-the-fly determinization. The key ideas are the usage of boxes and the composition operator.

The boxes can be used to determinize the automaton A. The corresponding deterministic automaton has the boxes of the transition monoid of A as states and the transitions are given by the composition operator ;. We call this automaton the box automaton of A. It is defined as follows.

Definition 24
The box automaton of A is given by $\mathcal{A}_M = (T, B(A), \rho_\varepsilon, B_F, \rightarrow_M)$. It has the following properties.

- *The set of states is given by the finite set of boxes$B(A)$ of A.*
- *The starting state is ρ_ε.*
- *The final states are given by the accepting boxes $B_F = \{\rho \mid (q_0, q_f) \in \rho$, where $q_F \in Q_F\}$ where q_0 is the starting state of A and Q_F the set of final states of A.*
- *The set of transitions is given by $\tau \xrightarrow{a} \tau; \rho_a$ for $\tau, \rho_a \in B(A)$ and $a \in T$.*
- *For each word $w \in T^*$ we have that $\rho_\varepsilon \xrightarrow{w}{}^*_M \rho_w$, where $\rightarrow^*_M$ is the reflexive and transitive closure of $\rightarrow_M$.*
- *The language of $\mathcal{A}_M$ coincides with the language of A: $\mathcal{L}(\mathcal{A}_M) = \mathcal{L}(A)$.*
- *The automaton $\mathcal{A}_M$ is deterministic.*

Figure 35 (left) depicts the box automaton of example automaton $\mathcal{A}_{ex}$ (Figure 34). Note that the box automaton of A may be strictly larger than the minimal deterministic automaton of A (Figure 35 (right)).

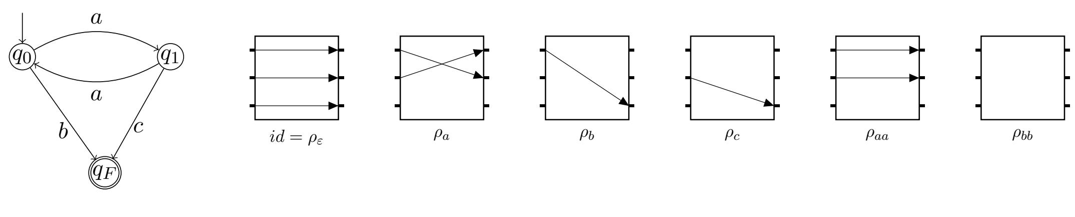


Figure 34.: Automaton $\mathcal{A}_{ex}$ and its boxes. The first dash represents state q_0, the second one q_1 and the third q_F. Box ρ_b is the only accepting box.

The problem with upfront determinization is that in most cases the automaton A can read more terminal words than the grammar G can produce. Consider for example the grammar $\mathcal{G}_{ex}$ from Section 3.

$$S_\square \rightarrow b \mid XY$$
$$X_\bigcirc \rightarrow a \mid aX$$
$$Y_\square \rightarrow b \mid c$$

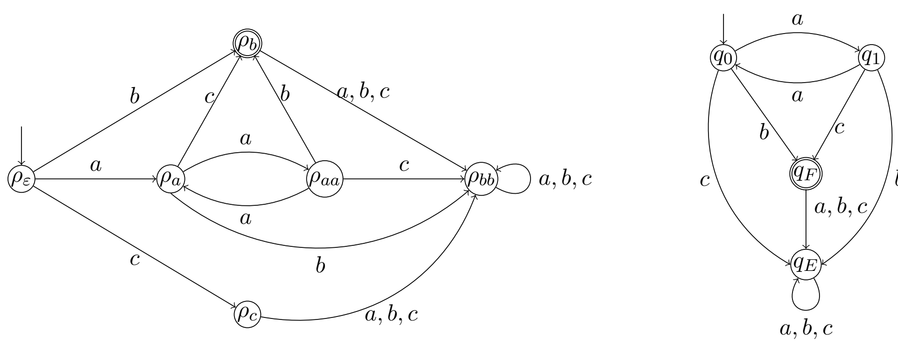


Figure 35.: Box graph of the automaton $\mathcal{A}_{ex}$ (left) and a minimal deterministic automaton recognizing $\mathcal{L}(A)$ (right).

We have that $\mathcal{L}(\mathcal{G}_{ex}) = a^*(b \mid c)$ if we ignore the ownership partition and regard $\mathcal{G}_{ex}$ as regular grammar. However, $\mathcal{A}_{ex}$ can read any word from $(a \mid b \mid c)^*$ and can decide whether it is accepted or not. If we determinize the whole automaton, we create unique paths for words which can not even by produced by the grammar.

In the summary algorithm, we are able to only compute the boxes of derivable words.

We start with the formulas $\sigma_a = \rho_a$ representing the plays starting from $a \in T$. The boxes of the terminals can easily be determined from the original automaton A without the need to determinize A. Then, we determine the words derivable from some non-terminal X by composing the formulas σ_{α_i} for $i = 1, \ldots, n$ if $X \to_G \alpha$ is a rule with left-hand side X. Note that we do not take the alternating structure of the plays into account and only focus on the derivable words. The key to this step is the ;-operator, which allows us to compose plays. By this method, for all terminal words w derivable from X, we get the state ρ_w that the box automaton would be in after reading w. Thus, we can determinize A on-the-fly along the words derivable from the starting symbol S, which may leave parts of the box automaton $\mathcal{A}_M$ untouched.

The other two approaches however do not get around a upfront determinization of A. We already discussed in the respective sections (Section 4 and 5) that we can not simply execute the algorithms with a non-deterministic automaton as the right-hand side of the inclusion. An on-the-fly determinization as in the summary approach is impossible as well as we explain below. Both the saturation and the Guess & Check algorithms need to know the complete deterministic automaton at the beginning of their computations.

Saturation approach To avoid an upfront determinization of A, we could adapt the alternating automaton to work with sets of states of A (macrostates) instead of states of the determinized automaton A^{det}. We denote the adapted alternating automaton by $\mathscr{A}'$. This adaptation simply mimics the powerset construction of the determinization and can compute the transitions of the original alternating automaton $\mathscr{A}$ on-the-fly. The transitions would be between a set of states of A and a set of sets of states of A:

$$q \xrightarrow{a}_{\mathscr{A}} Q$$

where q is a set of states of A and Q is a set of sets of states of A. The transitions can be derived from the transitions of A. The saturation rules are similar to the ones for $\mathscr{A}$. If we can identify the paths of $\mathscr{A}$ that do not correspond to words derivable in G, we can omit the computation of the corresponding transitions during the on-the-fly computations. However, the following discussion shows that this apporach fails.

The saturation method uses transitions to represent plays and paths to compose them instead of the ;- operator. These differences make an on-the-fly determinization impossible. As in the summary approach, we start with the plays starting from terminals a. They are represented by the transitions in the initial, unsaturated automaton $\mathscr{A}'_0$. Then, we determine the state changes induced by words derivable in a play from non-terminal X by considering paths

$$q_0 \xrightarrow{\alpha_1}_{\mathscr{A}'} Q_1 \xrightarrow{\alpha_2}_{\mathscr{A}'} \dots \xrightarrow{\alpha_n}_{\mathscr{A}'} Q_n$$

in $\mathscr{A}'$ (suppose we have rule $X \rightarrow_G \alpha_1 \dots \alpha_n$). This entails that at some point in the saturation, we had to add transitions with right-hand side $q \in Q_i$ for $i = 1, \dots, n-1$ (we already described this in Section 4). At this point, transition $q_0 \xrightarrow{\alpha_1}_{\mathscr{A}'} Q_1$ may not yet have existed. This means that we can not simply leave out some transitions of $\mathscr{A}$ during the saturation as we may need the transition in a succeeding saturation step. Therefore, the on-the-fly construction is bound to fail.

In general, the saturation approach results in "useless" paths in $\mathscr{A}$. On one hand paths $q_0 \xrightarrow{\alpha}{}^{*}_{\mathscr{A}} Q$ may exist s.t. α is not derivable from S. On the other hand, there may also be paths $q \xrightarrow{\alpha}{}^{*}_{\mathscr{A}} Q$ s.t. α is derivable from S, but the paths are never used in a saturation step starting from q_0. More precisely, this means that no path $q_0 \xrightarrow{\beta}{}^{*}_{\mathscr{A}} Q'$ with $q \in Q'$ exists in $\mathscr{A}$ s.t. $X \rightarrow_G \beta\alpha$ is a grammar rule. The chances that "useless" paths appear in $\mathscr{A}$ is actually very high. This is due to the fact that A^{det} is a deterministic automaton and thus is in particular complete i.e. has a transition from every state for every non-terminal.

Guess & Check approach A similar adaptation as for the saturation approach could be applied to the game graph $\mathcal{G}_{\mathrm{GC}}$ of the *GC game* in order to set up for an on-the-fly determinization. The state q of A^{det} that is stored in each vertex of $\mathcal{G}_{\mathrm{GC}}$ could be replaced by a set of states of A and the predictions $P \subseteq 2^{A^{\mathrm{det}}}$ replaced by sets of sets of states of A. Again, this construction simply mimics the powerset construction of the determinization step, the macrostates correspond to states in A^{det}. The idea is to exclude certain states from A^{det} from the predictions, if we know that they cannot be reached by words derivable in G.

The Guess & Check approach differs inherently from the other approaches in the sense that it is a top-down approach. Both the summarization and the saturation approach compute the plays bottom-up. The computation begins with the plays starting from terminals and then uses the grammar rules backwards to compute the plays starting from non-terminals X i.e. if we have rule $X \rightarrow_G \alpha$, we derive the plays starting at X from the plays starting at α.

In the Guess & Check approach however, we use the grammar rules forwards. At the claim-nodes $\mathit{Claim}(X\gamma, P', P, q)$, refuter makes a prediction about the terminal words derivable from X in form of the state changes that they induce from q. If prover wants to verify the prediction P', grammar rules $X \rightarrow_G \alpha$ are applied and the computation continues by verifying the same prediction P' for α.

However at the claim-nodes, where the predictions are made the derivation process has not yet taken place and the derivable terminal words are unknown. This means that we can not safely omit states from A^{det} (or macrostates of A^{det}) and only allow refuter to use a subset of the states in her prediction. Thus, we need to know the whole state space of A^{det} for the predictions.

9. Comparing the computed information

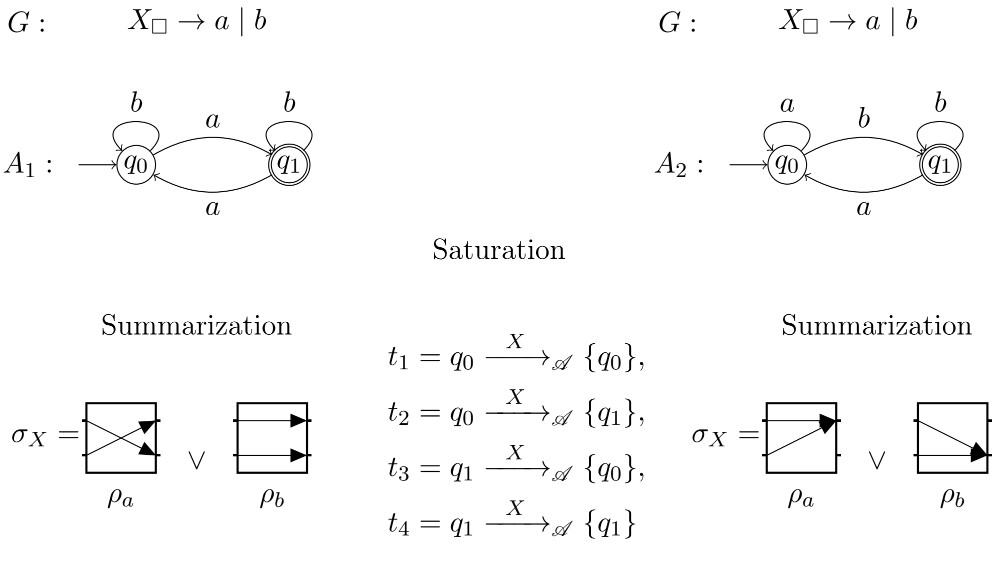
In this section, we compare the information which is computed during the fixed-point iterations in the three algorithms. We take the formulas $\sigma_X^{(k)}$, σ_X of the summary approach as base and determine whether they can be retrieved from the (intermediary) information computed by the saturation resp. Guess & Check method and vice-versa.

We first compare the summary approach to the saturation approach. The information that is computed during the saturation, are the newly added transitions to the alternating automaton $\mathscr{A}$. The saturation process starts with the alternating automaton $\mathscr{A}_0$ containing only transitions for terminals. Each saturation step creates a new alternating automaton $\mathscr{A}_i$ by adding new transitions to $\mathscr{A}_{i-1}$. We pair each of the newly added transitions with a time stamp t, indicating in which of the iterations they were added to $\mathscr{A}$. We show how the transitions with time stamp $t \leq k$ relate to the formulas $\sigma_X^{(k)}$.

In the second part of the section, we compare the summary approach to the Guess & Check method. The fixed-point iteration in the Guess & Check method corresponds to the antichain-attractor construction. In the first iteration, the attractor set $\overline{\mathsf{Attr}}^{(0)}_{\mathbb{A},\bigcirc}$ corresponds to the set of test-nodes $\mathit{Test}(q, P)$ that are winning for refuter i.e. where $q \in P$. Then, attractor set $\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ is computed from $\overline{\mathsf{Attr}}^{(k-1)}_{\mathbb{A},\bigcirc}$ by adding appropriate predecessors of the nodes in $\overline{\mathsf{Attr}}^{(k-1)}_{\mathbb{A},\bigcirc}$. We relate the nodes contained in $\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ to the formulas $\sigma_X^{(k)}$.

9.1. Summarization approach vs. Saturation approach

At the first glance, it does not seem to be possible to extract the intermediary formulas $\sigma_X^{(k)}$ from the transitions of the saturated alternating automaton $\mathscr{A}$. The problem lies in the different preprocessing steps of the two algorithms, which are conducted before the fixed-point iteration.



$G: \quad X_\square \to a \mid b$
$G: \quad X_\square \to a \mid b$
$A_1:$
$A_2:$
Saturation
Summarization
Summarization
$t_1 = q_0 \xrightarrow{X}_{\mathscr{A}} \{q_0\}$,
$t_2 = q_0 \xrightarrow{X}_{\mathscr{A}} \{q_1\}$,
$t_3 = q_1 \xrightarrow{X}_{\mathscr{A}} \{q_0\}$,
$t_4 = q_1 \xrightarrow{X}_{\mathscr{A}} \{q_1\}$
$\sigma_X =$
ρ_a
ρ_b
$\sigma_X =$
ρ_a
ρ_b

Figure 36.: Reconstruction of the boxes fails. Two context-free games with their formulas (left/right) and the transitions of $\mathscr{A}$ (middle). Although the formulas differ, the the transitions are the same for both games.

The summary algorithm bundles transitions of the automaton A together to form boxes and during the Kleene iteration, we only compute on the boxes, not the single transitions in A. In the saturation algorithm however, the transitions of A are directly translated to transitions in the alternating automaton $\mathscr{A}$ and during the saturation they are treated individually (even if they are labeled with the same terminal).

In order to retrieve the formulas from the saturated automaton, we would have to assemble the boxes from the transitions. The following example (Figure 36) shows that this is in general not possible.

In the example, we have two context-free games $\mathcal{G}_1$ resp. $\mathcal{G}_2$ given by grammar G and deterministic automaton A_1 resp. A_2. For both games, the saturation process leads to exactly the same transitions, depicted in the middle of the figure. Let us try to determine the formula σ_X from the transitions. In Section 4, we explained that the choices of prover are represented by sets of states and the choices of refuter by non-determinism. For both states q_0, q_1, we have two transitions labeled by X leading to a macrostate containing only one element. Thus, intuitively the formula should be of shape $\tau \vee \rho$ for some boxes τ, ρ. It remains to assemble the boxes τ and ρ from the state changes indicated by the transitions. However, the construction fails at this step as we do not know how to match the transitions. Pairing t_1 with t_4 and t_2 with t_3 will lead to the formula on the left, which corresponds to $\mathcal{G}_1$. On the other hand, pairing t_1 with t_3 and t_2 with t_4 produces the formula on the right side of Figure 36, corresponding to $\mathcal{G}_2$. Thus, there is a correct way to assemble the transitions to get the formulas, but we can not determine which one.

Thus, this difference in the preprocessing of the algorithms does not allow to retrieve the formulas from the transitions. However, the fact that there is a way to correctly assemble the transitions to construct the formulas, hints that there is a relation between the information computed during the fixed-point iterations in the two algorithms. Thus, our formulas do not seem to be a good base to compare the saturation and the summary approach. Luckily, the summary paradigm is not limited to the algorithm that we presented in Section 3. In [8], an other summary algorithm was presented, which better matches the mechanics of the saturation approach. It is also based on a Kleene iteration to determine formulas based on a system of equations. As for the summary algorithm from Section 3, the formulas summarize plays, although in a different way.

The rest of this subsection is organized as follows. First, we introduce the alternative summary algorithm. Then, we show that the formulas computed by the alternative algorithm can be determined from the transitions of the alternating automaton. Finally, we will come back to the summary algorithm from Section 3 and show that under the right circumstances, the formulas can be derived from the transitions.

9.1.1. Alternative Summary Algorithm

Let us recall the idea behind the formulas in the summary algorithm from Section 3. Formula σ_X captures the plays starting from X by interpreting provers choices by $\wedge$ and refuters choices by $\vee$. The atomic propositions are the boxes of the words derivable from X.

For the alternative algorithm, we slightly change this idea. In addition to the non-terminal X, we also fix a state q of A. Then, the formula $\sigma_{q,X}$ captures plays starting from X from state q on. The choices of prover resp. prover are again handled by $\wedge$ resp. $\vee$. But instead of the boxes of the derivable terminal words w, the atomic proposition represent the target state p of the run of A on w from q on, i.e. $q \xrightarrow{w} p$. We emphasize the singular of the word "run" in the previous sentence. This summary-based algorithm does not get around a upfront determinization of A. The reason is the same as for the saturation and Guess & Check approach. We can not let any of the players resolve the non-determinism and therefore have to work with a deterministic automaton on the right-hand side of the inclusion.

Because we do not mention the (non-determinized) automaton A anymore, assume that the starting state of A^{det} is given by q_0, the set of states by Q, the transitions by $q \xrightarrow{a} p$ and the final states by Q_F for the rest of this section.

The formulas are computed in the same manner as in the other summary-based algorithm, by a Kleene iteration on a system of equations derived from the rules of the grammar. Our domain consists again of the positive Boolean formulas, but this time over the states of A^{det} augmented by the unsatisfiable formula *false* modulo logical equivalence. The partial ordering is implication $\Rightarrow$ as in Section 3. We use the formula *false* to handle infinite plays and we factorize by logical equivalence $\Leftrightarrow$ to make the partial ordering $\Rightarrow$ antisymmetric on the domain.

The equation for each terminal a and state q of A^{det} is given by

$$\Delta_{q,a} = p \text{ if } q \xrightarrow{a} p.$$

The equation for each non-terminal X and state q of A^{det} is defined as follows.

$$\Delta_{q,X} = \bigotimes_{X \to \alpha_1 \dots \alpha_n} \Delta_{q,\alpha_1} : \Delta_{Q,\alpha_2} \cdots : \Delta_{Q,\alpha_n}, \text{ where } \bigotimes = \begin{cases} \vee & \text{if } X \in \bigcirc \\ \wedge & \text{if } X \in \Box \end{cases}$$

Note that we use an other operator :, called the matching operator, to compose the formulas for $\alpha_1, \dots, \alpha_n$ and that we use the set of states Q in the notation Δ_{Q,α_i} for $i = 2, \dots, n$ instead of a single state p. The notation Δ_{Q,α_i} actually represent a family of formulas $(\Delta_{p,\alpha_i})_{p \in Q}$ instead of a single formula. To understand this, recall the structure of a play from sentential form XY. First, vY is derived for some $v \in T^*$ in the play from XY and then the play carries on to derive $vw \in T^*$ from vY. Suppose we want to compute the target states in A from q on of terminal words vw derivable in plays from XY. We call these the plays from XY and q on. If we translate this setting to variables, we have $\Delta_{q,X} : \Delta_{Q,Y}$. The variable $\Delta_{q,X}$ is a placeholder for the plays starting from X and q. At this moment, the target states induced by the terminal words v derivable from X are yet unknown. Thus, we do not know from which state the plays from Y should start. We can only narrow it down to the family of formulas $\Delta_{Q,Y}$. But as soon as the variable $\Delta_{q,X}$ is replaced by the summary of the plays, we can resolve the uncertainty by matching $\Delta_{Q,Y}$ with each of the plays from X and q on. If the terminal word v derived during this play induces target state q' from q, we can resolve $\Delta_{Q,Y}$ to $\Delta_{q',Y}$.

The discussion leads to the following definition of the operators. G, H represent formulas, F_Q a family of formulas and q, p atomic propositions.

$$(G \bigotimes H) : F_Q = G : F_Q \bigotimes H : F_Q \qquad\qquad q : F_Q = F_q$$

where $\bigotimes \in \{\wedge, \vee\}$. The formula *false* is handled on the syntactic level by *false* $: F_Q =$ *false*. Lemma 11 in [MMN] shows that the :-operator is associative. Thus, we have

$$G : (F_Q : F'_Q) = (G : F_Q) : F' : Q = G : F_Q : F'_Q.$$

Clearly, the operators $\wedge, \vee$ are monotonic and Lemma 11 in [MMN] shows that the same holds for operator :. Thus, by the same line of argumentation as in Section 3, we get that the Kleene iteration computes a least solution for the system of equations. For each non-terminal X and state q of A^{det}, we denote the intermediary solutions of the Kleene iteration by $\sigma^{(k)}_{q,X}$ and the least solution by $\sigma_{q,X}$. As for the formulas of the regular summary algorithm, the solutions $\sigma^{(k)}_{q,X}$ capture all plays where at most k rules are applied in each branch of the parse tree corresponding to the play.

To determine the winner of the game, we proceed as in section 3. We are interested in the formula $\sigma_{q_0,S}$ where q_0 is the starting state of A^{det} and S the starting symbol of G. We define an assignment for the atomic propositions and evaluate the formula. As we play from the point of view of refuter, we assign *true* to all states that are non-final and *false* to the states that are accepting. If $\sigma_{q_0,S}$ evaluates to *true*, refuter wins, otherwise prover does.

9.1.2. Example of the alternative summary algorithm

To promote a better understanding of the new algorithm, we present an example of the computations. Let us revisit the running example of Section 3 (Figure 37).

$$\begin{aligned} S_{\square} &\to c \mid XY \\ X_{\bigcirc} &\to a \mid aX \\ Y_{\square} &\to b \mid c \end{aligned}$$

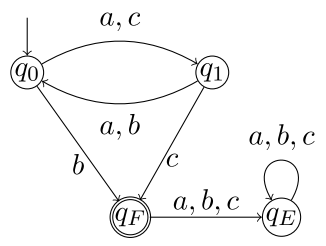


Figure 37.: Grammar $\mathcal{G}_{ex}$ and (determinized) automaton $\mathcal{A}_{ex}$ from Section 3.

We determinized the automaton $\mathcal{A}_{ex}$ by adding an error state q_E with a self-loop for all terminals a, b, c. From $\mathcal{G}_{ex}$ and $\mathcal{A}_{ex}$, we derive the following system of equations

$$\forall q \in \{q_0, q_1, q_F, q_E\} : \begin{cases} \Delta_{q,S} = \Delta_{q,c} \wedge \Delta_{q,a} : \Delta_{Q,X} : \Delta_{Q,Y} \\ \Delta_{q,X} = \Delta_{q,a} \vee \Delta_{q,a} : \Delta_{Q,X} \\ \Delta_{q,Y} = \Delta_{q,b} \wedge \Delta_{q,c} \end{cases}$$

For the sake of readability, we only consider a subset of the variables for the Kleene iteration:

$$\begin{aligned} &\Delta_{q_0,S} = \Delta_{q_0,c} \wedge \Delta_{q_0,a} : \Delta_{Q,X} : \Delta_{Q,Y} = q_1 \wedge q_1 : \Delta_{Q,X} : \Delta_{Q,Y} = q_1 \wedge \Delta_{q_1,X} : \Delta_{Q,Y} \\ &\Delta_{q_0,X} = \Delta_{q_0,a} \vee \Delta_{q_0,a} : \Delta_{Q,X} = q_1 \vee \Delta_{q_1,X} \\ &\Delta_{q_1,X} = \Delta_{q_1,a} \vee \Delta_{q_1,a} : \Delta_{Q,X} = q_0 \vee \Delta_{q_0,X} \\ &\Delta_{q_0,Y} = \Delta_{q_0,b} \wedge \Delta_{q_0,c} = q_F \wedge q_1 \\ &\Delta_{q_1,Y} = \Delta_{q_1,b} \wedge \Delta_{q_1,c} = q_0 \wedge q_F \end{aligned}$$

All other solutions are computed analogously.

i	$\sigma^{(i)}_{q_0,S}$
0	*false*
1	$q_1 \wedge \mathit{false} : \Delta_{Q,Y} = \mathit{false}$
2	$q_1 \wedge q_0 : \Delta_{Q,Y} = q_1 \wedge \Delta_{q_0,Y} = q_1 \wedge q_1 \wedge q_F$
3	$q_1 \wedge (q_0 \vee q_1) : \Delta_{Q,Y} = q_1 \wedge (\Delta_{q_0,Y} \vee \Delta_{q_1,Y}) = q_1 \wedge ((q_1 \wedge q_F) \vee (q_0 \wedge q_F)$

i	$\sigma^{(i)}_{q_0,S}$	$\sigma^{(i)}_{q_0,X}$	$\sigma^{(i)}_{q_1,X}$	$\sigma^{(i)}_{q_0,Y}$	$\sigma^{(i)}_{q_1,Y}$
0	*false*	*false*	*false*	*false*	*false*
1	*false*	q_1	q_0	$q_1 \wedge q_F$	$q_0 \wedge q_F$
2	$q_1 \wedge q_0$	$q_1 \vee q_0$	$q_0 \vee q_1$	$q_1 \wedge q_F$	$q_0 \wedge q_F$
3	$q_1 \wedge (q_0 \vee q_1)$	$q_1 \vee q_0$	$q_0 \vee q_1$	$q_1 \wedge q_F$	$q_0 \wedge q_F$

If we set q_F to *false* and all other atomic propositions to *true* in $\sigma_{q_0,S} = \sigma^{(3)}_{q_0,S} = q_1 \wedge ((q_1 \wedge q_F) \vee (q_0 \wedge q_F)$, the formula evaluates to *false* as expected. Indeed, recall from Section 3 that prover wins this context-free game.

Note that we are actually only interested in the solution of $\Delta_{q_0,S}$. All other solutions are only used to compute the value $\sigma_{q_0,S}$. In the Kleene iteration above, we only considered the variables that were needed, however in general the algorithm can not decide which variables are needed and which are superfluous. Therefore, simply all solutions $\sigma_{q,X}$ are computed.

9.1.3. Set representation of the formulas

The goal of the comparison is to determine whether the formulas $\sigma_{q,X}$ can be retrieved from the transitions of the saturated alternating automaton $\mathscr{A}$. The transitions are of shape

$$\begin{gathered} q \xrightarrow{X}_{\mathscr{A}} P_1 \\ \vdots \\ q \xrightarrow{X}_{\mathscr{A}} P_n \end{gathered}$$

where $P_1, \dots, P_n$ are sets of states of A^{det}. Thus, we would have to compare a set of transitions, whose right-hand sides are sets of states to a Boolean formula. Clearly, these two representations do not match very well, making the comparison more involved. Fortunately, it is possible to represent Boolean formulas by sets as well. After bringing the Boolean formula into disjunctive normal form (DNF), we can represent the clauses by sets of atomic propositions and the whole formula as a set of sets. Clearly, the two representations are equivalent.

We represent the unsatisfiable formula *false* by the empty set $\{\}$.

In the example above, the DNF of $\sigma_{q_0,S}$ is given by

$$\sigma_{q_0,S} = (q_1 \wedge q_F) \vee (q_1 \wedge q_0 \wedge q_F).$$

Thus, the set representation of the formula is

$$\sigma_{q_0,S} = \{\{q_1, q_F\}, \{q_0, q_1, q_F\}\}.$$

We are interested in the set representations of the (intermediary) formulas $\sigma^{(k)}_{q,X}$. In Boolean formula representation, they are given by

$$\sigma^{(k)}_{q,X} = \bigotimes_{X \to \alpha_1 \dots \alpha_n} \sigma^{(k-1)}_{q,\alpha_1} : \sigma^{(k-1)}_{Q,\alpha_2} \cdots : \sigma^{(k-1)}_{Q,\alpha_n}, \text{ where } \bigotimes = \begin{cases} \vee & \text{if } X \in \bigcirc \\ \wedge & \text{if } X \in \square \end{cases}$$

where $\sigma^{(k-1)}_{Q,\alpha_i}$ stands for the family $(\sigma^{(k-1)}_{q,\alpha_i})_{q \in Q}$.

Assuming that $\sigma^{(k-1)}_{q,\alpha_i}$ are already in set representation for $i = 1, \dots, n$ and all $q \in Q$, we have to define how the operations :, $\wedge$ and $\vee$ combine the sets and prove that the resulting set is as expected.

We first consider each operator on its own and then combine the results to obtain a set theoretic representation for $\sigma^{(k)}_{q,X}$.

Matching operator Let us start with a easy example to provide some intuition of the definition ahead. Let us consider $G : F_Q$ for formula G and family of formulas F_Q given by

$$\begin{aligned} G &= (q_1 \wedge q_2) \vee q_3 \\ F_{q_1} &= (p_1 \wedge p_2) \vee p_3 \\ F_{q_1} &= (p_4 \wedge p_5) \vee p_6 \\ F_{q_1} &= (p_7 \wedge p_8) \vee p_9. \end{aligned}$$

Then, $G : F_Q$ is given by

$$\begin{aligned} G : F_Q &= \big(\underbrace{(q_1 \wedge q_2)}_{K_1} \vee \underbrace{q_3}_{K_2}\big) : F_Q \\ &= (F_{q_1} \wedge F_{q_2}) \vee F_{q_3} \\ &= \underbrace{\Big[\big((p_1 \wedge p_2) \vee p_3\big) \wedge \big((p_4 \wedge p_5) \vee p_6\big)\Big]}_{H_1} \vee \underbrace{\Big[(p_7 \wedge p_8) \vee p_9\Big]}_{H_2} \end{aligned}$$

Note that we marked the parts of the formula by H_1 resp. H_2 that stem from clause K_1 resp. K_2 of G by a bracket. For the set representation, we need to bring the formula into DNF. We use distributivity to normalize the formula (Figure 38).

$$\begin{aligned} G : F_Q &= \Big[\big((p_1 \wedge p_2) \vee p_3\big) \wedge \big((p_4 \wedge p_5) \vee p_6\big)\Big] \vee \Big[(p_7 \wedge p_8) \vee p_9\Big] \\ &= \Big[(p_1 \wedge p_2 \wedge p_4 \wedge p_5) \vee (p_1 \wedge p_2 \wedge p_6) \vee (p_3 \wedge p_4 \wedge p_5) \vee (p_3 \wedge p_6)\Big] \vee \Big[(p_7 \wedge p_8) \vee p_9\Big] \end{aligned}$$

Figure 38.: Normalizing $G : F_Q$.

Let us point ou the steps in this computation that lead to for example clause $(p_1 \wedge p_2 \wedge p_3 \wedge p_4)$. In the first step, we replaced q_1 resp. q_2 in K_1 by F_{q_1} and F_{q_2}. In the resulting subformula H_1, we have two operator alternations. The outermost operator is the $\wedge$ from clause K_1 of G. Then, we have the alternation of $\vee$ and $\wedge$ of the formulas F_{q_1} resp. F_{q_2}. To reach DNF, we multiply every clause in F_{q_1} with every clause in F_{q_2}. For our example clause, we matched clause $(p_1 \wedge p_2)$ with $(p_3 \wedge p_4)$. In general, we can do the replacing of the atomic propositions by formulas and the clause matching in H_1 by a function

$$\begin{aligned} z : & K_1 \to \bigcup_{q \in Q} F_q \\ & q \mapsto F_q \end{aligned}$$

which maps every atomic proposition q in K_1 to a clause in F_q. By considering every possible function of this shape, we get every clause from H_1. For the example clause, we have function $q_1 \mapsto \{p_1, p_2\}$ and $q_2 \mapsto \{p_4, p_5\}$ if we assume that G and $(F_q)_{q \in Q}$ are already in set representation. We proceed analogously for K_2 and H_2. In this case, the function z only has to map q_3 to either $\{p_7, p_8\}$ or to $\{p_9\}$.

The previous discussion leads to the following definition.

$$G : F_Q = \bigcup_{K \in F} \bigcup_{\substack{z : K \to \bigcup_{q \in Q} F_q \\ q \mapsto F_q}} \{ \bigcup_{p \in K} z(p) \}$$

In the setting of the formulas $\sigma^{(k)}_{q,X}$, $G = G_{p_0}$ will be a member of a family of formulas

$(G_q)_{q \in Q}$. To use a more unified notation, we can rewrite the definition above to

$$G : F_Q = \bigcup_{\substack{z_1 : I_0 \to \bigcup\limits_{q \in Q} G_q \\ q \mapsto G_q}} \bigcup_{\substack{z_2 : I_1 \to \bigcup\limits_{q \in Q} F_q \\ q \mapsto F_q}} \{ \bigcup_{p_1 \in z_1(p_0)} z_2(p_1) \}$$

where $I_0 = \{p_0\}$ and $I_1 = img(z_1)$ is the image of function z_1.

In general, we will have compose a formula with several families. We will use the following notation for multiple successive compositions at once. Let $G_q^{(1)}$ be a formula and $G_Q^{(2)}, \dots, G_Q^{(n)}$ be families of formulas.

$$G_q^{(1)} \overset{n}{\underset{i=2}{\odot}} G_Q^{(i)} = G_q^{(1)} : G_Q^{(1)} : G_Q^{(2)} : \cdots : G_Q^{(n)}$$

The following theorem shows how we lift the definition above to multiple successive compositions and proves the correctness.

Theorem 4
Let $G_q^{(1)}$ be a formula and $G_Q^{(2)}, \dots, G_Q^{(n)}$ be families of formulas. Then, we have that

$$G_q^{(1)} \overset{n}{\underset{i=2}{\odot}} G_Q^{(i)} = \bigcup_{\substack{z_1 : I_0 \to \bigcup\limits_{p \in Q} G_p^{(1)} \\ p_0 \mapsto C_{p_0}^{(1)}}} \bigcup_{\substack{z_2 : I_1 \to \bigcup\limits_{p \in Q} G_p^{(2)} \\ p_1 \mapsto C_{p_1}^{(2)}}} \cdots \bigcup_{\substack{z_n : I_{n-1} \to \bigcup\limits_{p \in Q} G_p^{(n)} \\ p_{n-1} \mapsto C_{p_{n-1}}^{(n)}}} \left\{ \bigcup_{p_1 \in z_1(p_0)} \bigcup_{p_2 \in z_2(p_1)} \cdots \bigcup_{p_{n-1} \in z_{n-1}(p_{n-2})} z_n(p_{n-1}) \right\}$$

where

- $I_0 = \{q\}$
- $I_i = img(z_i)$ *are the images of the functions* z_i *for* $i = 1, \dots, n$,
- $C_{p_{j-1}}^{(i)} \in G_{p_{i-1}}^{(i)}$ *are clauses of the formulas* $G_{p_{i-1}}^{(i)}$ *for* $i = 1, \dots, n$ *and*
- $p_i \in I_i$, *for* $i = 1, \dots, n$.

Before we prove this theorem, we introduce a helping lemma. Let therefore be $z_1, \dots, z_n$ s.t.

$$z_i : I_{i-1} \to \bigcup_{p \in Q} G_q$$
$$p_{i-1} \mapsto C_{p_{i-1}}^{(i)} \in G_{q_{i-1}}.$$

Lemma 15

1. *For* $i = 1, \dots, n$, *it holds that* $I_i = \bigcup\limits_{p \in I_{i-1}} z_i(p)$.

2. $I_1 = z_1(\rho_0)$
   $I_i = \bigcup\limits_{p_1 \in z_1(p_0)} \bigcup\limits_{p_2 \in z_2(p_1)} \cdots \bigcup\limits_{p_{i-1} \in z_{i-1}(p_{i-2})} z_i(p_{i-1})$ *for* $i = 2, \dots, n$

Proof. The first claim is clear by definition of the z_i.

We prove the second claim by induction over i.

Base case: $i = 1$

Then we have that

$$I_1 \overset{\text{Claim 1}}{=} \bigcup_{p_0 \in I_0} z_1(p_0) = z_1(p_0)$$

Induction step: $i \to i+1$

$$I_{i+1} \overset{\text{Claim 1}}{=} \bigcup_{p_i \in I_i} z_{i+1}(p_i).$$

Now, we use Claim 1 again to get that $I_i = \bigcup\limits_{p_{i-1} \in I_{i-1}} z_i(p_{i-1})$. Furthermore we use the following set-theoretic equation, which is easy to see.

$$\bigcup_{e \in \{\bigcup\limits_{f \in N} M_f\}} L_e \quad = \quad \bigcup_{f \in N} \; \bigcup_{e \in M_f} L_e, \text{ for } N, M_f, L_e \text{ sets.}$$

Then it follows that

$$I_{i+1} = \bigcup_{p_{i-1} \in I_{i-1}} \; \bigcup_{p_i \in z_i(p_{i-1})} z_{i+1}(p_i).$$

By using Lemma the first claim on I_{i-1}, we can apply this step iteratively until $i = 1$, where we can use that $I_1 = z_1(p_0)$. The claim follows.

$\square$

With the helping lemma at hand, we can prove Theorem 4.

Proof of Theorem 4 We prove the claim by induction over n.

Base case: $n = 2$

For the matching of a formula with a single family of formulas, the claim has been proven in [8].

Induction step: $n \to n+1$

Using the base case, we can split off the last family of formulas $G_Q^{(n+1)}$ and then apply the induction hypothesis for $n = 2$.

$$\begin{aligned}
G_q^{(1)} \overset{n+1}{\underset{i=2}{\bigodot}} G_Q^{(i)} &= \left(G_q^{(1)} \overset{n}{\underset{i=2}{\bigodot}} G_Q^{(i)} \right) : G_Q^{(n+1)} \\
&= G_q : G_Q^{(n+1)} \\
&= \bigcup_{\substack{z' : I_0 \to \bigcup\limits_{p \in Q} G_p \\ p_0 \mapsto C_{p_0}}} \; \bigcup_{\substack{z'_{n+1} : I \to \bigcup\limits_{p \in Q} G_p^{(n+1)} \\ p \mapsto C_p^{(n+1)}}} \left\{ \bigcup_{p \in z'(\rho_0)} z'_{n+1}(p) \right\} =: H'
\end{aligned}$$

where $I = img(z')$, $I_{n+1} = img(z'_{n+1})$ and $C_{p_0} \in G_{p_0}$, $C_p^{(n+1)} \in G_p^{(n+1)}$.

Let furthermore be

$$\begin{aligned}
H := & \bigcup_{\substack{z_1 : I_0 \to \bigcup\limits_{p \in Q} G_p^{(1)} \\ p_0 \mapsto C_{p_0}^{(1)}}} \cdots \bigcup_{\substack{z_n : I_{n-1} \to \bigcup\limits_{p \in Q} G_p^{(n)} \\ p_{n-1} \mapsto C_{p_{n-1}}^{(n)}}} \; \bigcup_{\substack{z_{n+1} : I_n \to \bigcup\limits_{p \in Q} G_p^{(n+1)} \\ p_n \mapsto C_{pn}^{(n+1)}}} \\
& \left\{ \bigcup_{p_1 \in z_1(p_0)} \cdots \bigcup_{p_{n-1} \in z_{n-1}(p_{n-2})} \; \bigcup_{p_n \in z_n(p_{n-1})} z_{n+1}(p_n) \right\}
\end{aligned}$$

the expected representation of $G^{(1)}_q \overset{n+1}{\underset{i=2}{\odot}} G^{(i)}_Q$.

Claim: $H' = H$

Proof. We prove the claim by mutual inclusion.

- "$\subseteq$"

  Let $K' \in H'$. Then, we know that there exist $z' : I_0 \to \bigcup_{p \in Q} G_p, p_0 \mapsto C_{p_0}$ and $z'_{n+1} : I \to \bigcup_{p \in Q} G^{(n+1)}_p, p \mapsto C^{(n+1)}_p$ s.t.

  $$K' = \{ \bigcup_{p \in z'(p_0)} z'_{n+1}(p_n)\}$$

  To show that $K' \in H$, we provide $z_1, \dots, z_{n+1}$ with $z_i : I_{i-1} \to \bigcup_{p \in Q} G^{(i)}_p, p_{i-1} \mapsto C^{(i)}_{p_{i-1}}$ s.t.

  $$K' = \{ \bigcup_{p_1 \in z_1(p_0)} \dots \bigcup_{p_{n-1} \in z_{n-1}(p_{n-2})} \bigcup_{p_n \in z_n(p_{n-1})} z_{n+1}(p_n)\} \in H$$

  By definition of z', we have that $I = z'(q) \in G_q$. Then, we can use the induction hypothesis, to write I as

  $$I = \{ \bigcup_{p_1 \in z_1(p_0)} \dots \bigcup_{p_{n-1} \in z_{n-1}(p_{n-2})} z_n(p_{n-1})\}$$

  for some $z_1, \dots, z_n$.

  Let us take $z_1, \dots, z_n$ as above. Then we have $I_n = I$ by Claim 1 of Lemma 15 and thus we can set $z_{n+1} = z'_{n+1}$. Using the claims of Lemma 15, this gives us

  $$\begin{aligned}
  &\{ \bigcup_{p_1 \in z_1(p_0)} \dots \bigcup_{p_{n-1} \in z_{n-1}(p_{n-2})} \bigcup_{p_n \in z_n(p_{n-1})} z_{n+1}(p_n)\} \\
  &\overset{\text{Claim 2}}{=} I_{n+1} \\
  &\overset{\text{Claim 1}}{=} \{ \bigcup_{p_n \in I_n} z_{n+1}(p_n)\} \\
  &= \{ \bigcup_{p_n \in I = z'(p_0)} z'_{n+1}(p_n)\} = K'
  \end{aligned}$$

- "$\supseteq$"

  Let $K \in H$. Therefore, there exist $z_1, \dots, z_{n+1}$ with $z_i : I_{i-1} \to \bigcup_{p \in Q} G^{(i)}_p, p_{i-1} \mapsto C^{(i)}_{p_{i-1}}$ s.t.

  $$K = \{ \bigcup_{p_1 \in z_1(p_0)} \dots \bigcup_{p_{n-1} \in z_{n-1}(p_{n-2})} \bigcup_{p_n \in z_n(p_{n-1})} z_{n+1}(p_n)\}.$$

  This time, we have to provide some $z' : I_0 \to \bigcup_{p \in Q} G_p, p_0 \mapsto C_{p_0}$ and $z'_{n+1} : I \to$

$\bigcup_{p\in Q} G_p^{(n+1)}, p \mapsto C_p^{(n+1)}$ s.t.

$$K = \{ \bigcup_{p\in z'(p_0)} z'_{n+1}(p_n)\}$$

Consider therefore

$$I_n = \{ \bigcup_{p_1\in z_1(p_0)} \cdots \bigcup_{p_{n-1}\in z_{n-1}(p_{n-2})} z_n(p_{n-1})\}.$$

By construction, we know that $I_n \in G$.

We set $z'(p_0) = I_n$, which entails $I = I_n$. Thus, we can fix $z'_{n+1} = z_{n+1}$, which is again well-defined. Finally, we have that

$$\{ \bigcup_{p\in z'(p_0)} z'_{n+1}(p_n)\} = \{ \bigcup_{p\in I=I_n} z_{n+1}(p_n)\} \overset{\text{Claim 1}}{=} I_{n+1} = K$$

by Lemma 15 which concludes the proof of the claim and the theorem.

□

Disjunction and Conjunction For the set theoretic representation of the $\wedge$- and $\vee$-operator, we refer to Lemma 18 of [8]. The authors consider conjunctive normal form, therefore we have to exchange the result for $\wedge$- and $\vee$. Let G, H be formulas in set representation in DNF, the we have that

$$G \vee H = G \cup H \qquad\qquad G \wedge H = \{K_1 \cup K_2 \mid K_1 \in G, K_2 \in H\}.$$

We can easily lift the definitions to the conjunction/disjunction of several formulas by induction. Let $G_1, \ldots, G_n$ be formulas in set representation.

$$\bigvee_{i=1}^{n} G_i = \bigcup_{i=1}^{n} G_i \qquad\qquad \bigwedge_{i=1}^{n} G_i = \{\bigcup_{i=1}^{n} K_i \mid K_i \in G_i\}.$$

We now have defined how the operators :, $\vee$ and $\wedge$ behave when applied to sets of states. This allows us to determine the set theoretic representation of the formulas $\sigma_{q,X}^{(k)}$. Recall that they were given by

$$\sigma_{q,X}^{(k+1)} = \bigotimes_{X\to\alpha_1\ldots\alpha_n} \sigma_{q,\alpha_1}^{(k)} : \sigma_{Q,\alpha_2}^{(k)} \cdots : \sigma_{Q,\alpha_n}^{(k)}, \text{ where } \bigotimes = \begin{cases} \vee & \text{if } X \in \bigcirc \\ \wedge & \text{if } X \in \Box \end{cases}$$

if represented as a Boolean formula.

Definition 25
Let now be

- $X \to_G \gamma$ *be the grammar rules with left-hand side* X*,*
- $n = |\gamma|$ *the size of the right-hand side of the grammar rule* $X \to_G \gamma$*,*
- $I_0^{(\gamma)} = \{q\}$ *for each rule* $X \to_G \gamma$*,*
- $I_j^{(\gamma)} = img(z_i^{(\gamma)})$*, for* $i = 1, \ldots, n$ *are the images of some functions* $z_i^{(\gamma)}$*,*
- $C_{p_{i-1}}^{(\gamma_i)} \in \sigma_{p_{i-1},\gamma_i}^{(k)}$*, for* $i = 1, \ldots, n$ *are the clauses of the formulas* $\sigma_{p_{i-1},\gamma_i}^{(k)}$*,*

- *and $p_i \in I_i^{(\gamma)}$, for $i = 1, \dots, n$.*

If we have now that $X \in \bigcirc$, the set-theoretic representation is given by

$$\sigma^{(k+1)}_{qX} = \bigcup_{X \to \gamma} \quad \bigcup_{\substack{z_1 : I_0 \to \bigcup_{p \in Q} \sigma^{(k)}_{p,\gamma_1} \\ p_0 \mapsto C^{(\gamma_1)}_{p_0}}} \quad \bigcup_{\substack{z_2 : I_1 \to \bigcup_{p \in Q} \sigma^{(k)}_{p,\gamma_2} \\ p_1 \mapsto C^{(\gamma_2)}_{p_1}}} \cdots \bigcup_{\substack{z_n : I_{n-1} \to \bigcup_{p \in Q} \sigma^{(k)}_{p,\gamma_n} \\ p_{n-1} \mapsto C^{(\gamma_n)}_{p_{n-1}}}}$$
$$\left\{ \bigcup_{p_1 \in z_1^{(\gamma)}(p_0)} \quad \bigcup_{p_2 \in z_2^{(\gamma)}(p_1)} \cdots \bigcup_{p_{n-1} \in z_{n-1}^{(\gamma)}(p_{n-2})} z_n^{(\gamma)}(p_{n-1}) \right\}$$

In case $X \in \Box$, we have

$$\sigma^{(k+1)}_{q,X} = \left\{ \bigcup_{X \to_G \gamma} K^{(\gamma)} \mid \right.$$
$$K^{(\gamma)} \in \bigcup_{\substack{z_1^{(\gamma)} : I_0^{(\gamma)} \to \bigcup_{p \in Q} \sigma^{(k)}_{p,\gamma_1} \\ p_0 \mapsto C^{(\gamma_1)}_{p_0}}} \quad \bigcup_{\substack{z_2^{(\gamma)} : I_1^{(\gamma)} \to \bigcup_{p \in Q} \sigma^{(k)}_{p,\gamma_2} \\ p_1 \mapsto C^{(\gamma_2)}_{p_1}}} \cdots \bigcup_{\substack{z_n^{(\gamma)} : I_{n-1}^{(\gamma)} \to \bigcup_{p \in Q} \sigma^{(k)}_{p,\gamma_n} \\ p_{n-1} \mapsto C^{(\gamma_n)}_{p_{n-1}}}}$$
$$\left. \left\{ \bigcup_{p_1 \in z_1^{(\gamma)}(p_0)} \quad \bigcup_{p_2 \in z_2^{(\gamma)}(p_1)} \cdots \bigcup_{p_{n-1} \in z_{n-1}^{(\gamma)}(p_{n-2})} z_n^{(\gamma)}(p_{n-1}) \right\} \right\}.$$

9.1.4. Comparison: Summarization vs. Saturation

With the set-theoretic representation of the formulas $\sigma^{(k)}_{q,X}$ at hand, we are now able to compare the information computed during the fixed-point iterations in the two approaches. More precisely, we want to relate the solutions $\sigma^{(k)}_{q,X}$ with the transitions $q \xrightarrow{X}_{\mathscr{A}} P$ of the alternating automaton $\mathscr{A}$ with left-hand side q and label X that appeared during the k-th saturation step in $\mathscr{A}_k$. To this end, we will add a time stamp to each transition to indicate the saturation step in which it was added to $\mathscr{A}$, i.e. $q \xrightarrow[k]{X}_{\mathscr{A}} P$ if the transition appeared in the k-th saturation step.

Comparing the formulas with the transitions leads to the following theorem.

Theorem 5
Let $q \in Q$ and $\alpha \in \mathscr{S}$. Then we have

$$\sigma^{(k)}_{q,\alpha} = \{K_1, \dots, K_m\} \iff q \xrightarrow[t \leq k]{\alpha}_{\mathscr{A}} K_j \textit{ for } j = 1, \dots, m.$$

Before we prove the theorem, we briefly explain the general idea behind the proof.

Proof idea Suppose we have $K \in \sigma^{(k)}_{q,X}$ and want to show that there exists a transition $q \xrightarrow[t \leq k]{X}_{\mathscr{A}} K$. From the set-theoretic representation of $\sigma^{(k)}_{q,X}$, we know that there exist appropriate functions $z_1, \dots, z_n$ defining clause K. On the other hand, transition $q \xrightarrow[t \leq k]{X}_{\mathscr{A}} K$ was added to $\mathscr{A}$ based on one of the saturation rules and (an) appropriate path(s) $q \xrightarrow[t<k]{\gamma}{}^{*}_{\mathscr{A}} K'$. We will use the functions $z_1, \dots, z_n$ to show the existence of such (a) path(s) in $\mathscr{A}$. More

precisely, they will define the intermediary positions of the path(s)

$$q \xrightarrow[t<k]{\gamma_1}_{\mathscr{A}} K^{(1)} \xrightarrow[t<k]{\gamma_2}_{\mathscr{A}} \dots K^{(n-1)} \xrightarrow[t<k]{\gamma_n}_{\mathscr{A}} K'.$$

For the other direction, we will use the intermediary positions to show the existence of appropriate functions $z_1, \dots, z_n$ defining clause K.

As we will need the intermediary positions of the paths in $\mathscr{A}$ for the proof, we unroll the definition of $\longrightarrow^*_{\mathscr{A}}$ as given in Section 4 to be able to reason about the intermediary positions of the paths.

Intermediary positions of the paths in the alternating automaton Suppose we have path $q \xrightarrow[t\leq k]{\gamma}{}^*_{\mathscr{A}} K$ in $\mathscr{A}$ for some $\gamma = \gamma_1, \dots, \gamma_n$, with $\gamma_i \in \mathscr{S}$ and $K \subseteq 2^Q$. Then, we define the intermediary positions $K^{(1)}, \dots, K^{(n)}$ of the path as follows.

Definition 26
Let $q \xrightarrow[t\leq k]{\gamma}{}^*_{\mathscr{A}} K$ be a path in $\mathscr{A}$. Then, we call $K^{(1)}, \dots, K^{(n)}$ with $K^{(i)} \subseteq 2^Q$ the intermediary positions of the path if

- *there exists a transition $q \xrightarrow[t\leq k]{\gamma_1}_{\mathscr{A}} K^{(1)}$ in $\mathscr{A}$.*
- *for $i = 1, \dots, n-1$, we have $K^{(i+1)} = \bigcup_{p \in K^{(i)}} D_p$, s.t. there exists a transition $p \xrightarrow[t\leq k]{\gamma_{i+1}}_{\mathscr{A}} D_p$ in $\mathscr{A}$.*
- *$K = K^{(n)}$*

Let us make an example for $|\gamma| = 2$. Suppose we have that $q \xrightarrow[t\leq k]{\gamma}{}^*_{\mathscr{A}} K$. Then, there exist $K^{(1)}, K^{(2)}$ s.t.

$$q \xrightarrow[t\leq k]{\gamma_1}_{\mathscr{A}} K^{(1)} = \begin{cases} p_1 \xrightarrow[t\leq k]{\gamma_2}_{\mathscr{A}} D_{p_1} \\ \vdots \\ p_k \xrightarrow[t\leq k]{\gamma_2}_{\mathscr{A}} D_{p_k} \end{cases}$$

and $K = K^{(2)} = \bigcup_{p_i \in K^{(1)}} D_{p_i}$.

Using the definition of $\longrightarrow^*_{\mathscr{A}}$, it is sufficient to show the existence of such $K^{(1)}, \dots, K^{(n)}$ to conclude the existence of the path $q \xrightarrow[t\leq k]{\gamma}{}^*_{\mathscr{A}} K^{(n)} = K$.

We will use the intermediary positions $K^{(1)}, \dots, K^{(n)}$ to define the functions $z_1, \dots, z_n$. For the other direction of the proof, we use the $z_1, \dots, z_n$ to show the existence of positions $K^{(1)}, \dots, K^{(n)}$ as defined above.

We split the proof into its two directions. The claim of Theorem 5 follows from Lemma 16 and Lemma 17.

Lemma 16
Let $\alpha \in \mathscr{S}$ and $q \in A^{det}$. Then, it holds that

$$\sigma^{(k)}_{q,\alpha} = \{K_1, \dots, K_m\} \Rightarrow q \xrightarrow[t\leq k]{\alpha}_{\mathscr{A}} K_j \text{ for } j = 1, \dots, m.$$

Proof. We prove the claim by induction over k.

Base case $\mathbf{k = 0}$:

We have to distinguish two cases: First, let $\alpha = a \in T$. Then, on one hand $\sigma^{(k)}_{q,X} = p$ if $q \xrightarrow{a} p$ by definition of the system of equations. On the other that $q \xrightarrow[0]{a}_{\mathscr{A}} p$ as the transitions of $\mathscr{A}_0$ correspond to the transitions in A^{det}. Before the first iteration we have not added any other transitions yet, meaning that this is the only such transition. Thus, the claim holds.

Now let $\alpha = X \in N$. Again by definition, $\sigma^{(k)}_{q,X} = \{\}$ (unsatisfiable formula *false*). As we have not started the saturation process yet, there is no transition labeled by a non-terminal X. Thus, the claim holds.

Induction step $\mathbf{k \to k+1}$: The case for $\alpha = a \in T$ is trivial. From $K \in \sigma^{(k+1)}_{q,a}$, it follows that $K = \{p\}$ if $q \xrightarrow{a} p$ in A^{det}. As all transitions from $\mathscr{A}_0$ correspond to transitions in A^{det}, we have that $q \xrightarrow[0]{a}_{\mathscr{A}} \{p\}$. The claim follows.

Let now $K \in \sigma^{(k+1)}_{q,X}$. Because the shape of the clause K differs depending on which player owns X, we need to distinguish two cases.

- $X \in \bigcirc$

  According to definition 25, there is a grammar rule $X \to_G \gamma$ s.t. we can write K as

$$K = \bigcup_{p_1 \in z_1(q)} \bigcup_{p_2 \in z_2(p_1)} \cdots \bigcup_{p_{n-1} \in z_{n-1}(p_{n-2})} z_n(p_{n-1})$$

  for some functions $z_1, \ldots, z_n$ ($n = |\gamma|$) with

$$\begin{aligned} z_i : & I_{i-1} \to \bigcup_{p \in Q} \sigma^{(k)}_{p,\gamma_i} \\ & p_{i-1} \mapsto C^{(\gamma_i)}_{p_{i-1}} \in \sigma^{(k)}_{p_{i-1},\gamma_j}. \end{aligned}$$

  Recall that $I_0 = \{q\}$ and $I_i = img(z_i)$ for $1 = 2, \ldots, n$.

  Overview: We use the functions z_i to show that there exists a path $q \xrightarrow[t \le k]{\gamma}{}^{*}_{\mathscr{A}} K$ for grammar rule $X \to_G \gamma$. Then, the claim follows from the first saturation rule.

  Let us thus define intermediary states $K^{(1)}, \ldots, K^{(n)}$ s.t.

  1. $K^{(1)} \in \sigma^{(k)}_{q,\gamma_1}$,
  2. for $i = 1, \ldots, n-1$ we have $K^{(i+1)} = \bigcup_{p \in K^{(i)}} D_p$ for some $D_p \in ; \sigma^{(k)}_{p,\gamma_i}$ and
  3. $K = K^{(n)}$.

  Then, it follows from the induction hypothesis and the first two points above that

  1. $q \xrightarrow[t \le k]{\gamma_1}_{\mathscr{A}} K^{(1)}$
  2. For $K^{(i+1)} = \bigcup_{p \in K^{(i)}} D_p$, there exists a transition $p \xrightarrow[t \le k]{\gamma_i}_{\mathscr{A}} D_p$ for each $p \in K^{(i)}$.

  From the existence of such $K^{(1)}, \ldots, K^{(n)}$, we can conclude that there is a path $q \xrightarrow[t \le k]{\gamma}{}^{*}_{\mathscr{A}} K^{(n)} \overset{3.}{=} K$

  As $X \in \bigcirc$, we get that $q \xrightarrow[t \le k]{X}_{\mathscr{A}} K$ by the first saturation rule, proving the claim.

Definition of $\mathbf{K^{(1)}}, \ldots, \mathbf{K^{(n)}}$: Let us define the $K^{(1)}, \ldots, K^{(n)}$ as introduced in 1., 2. and 3. above. We define the $K^{(s)}$ inductively for $i = 1 \ldots, n-1$ using the functions z_i. In order to ensure well-undefinedness of the functions z_i, we require that $K^{(i)} = I_i = img(z_i)$. Recall the results from Lemma 15, which we use to maintain the invariant $K^{(i)} = I_i$.

$$I_i = \bigcup_{p \in I_{i-1}} z_i(p) \text{ for } i = 1, \ldots, n$$
$$I_i = \bigcup_{p_1 \in z_1(p_0)} \bigcup_{p_2 \in z_2(p_1)} \cdots \bigcup_{p_{i-1} \in z_{i-1}(p_{i-2})} z_i(p_{i-1}) \text{ for } i = 2, \ldots, n$$

In particular, it holds that $I_n = K$.

First, we define that

$$K^{(1)} = z_1(q).$$

Then, $K^{(1)} \in \sigma^{(k)}_{q,\gamma_1}$ and $K^{(1)} = z_1(q) = \bigcup_{p \in I_0} z_1(p) = I_1$.

Let now $i \in \{1, \ldots, n-1\}$ and suppose $K^{(i+1)}$ has already been defined for $j = 1, \ldots, i$. We define

$$\begin{aligned} K^{(i+1)} &= \bigcup_{p \in K^{(i)}} D_p \\ &= \bigcup_{p \in K^{(i)}} z_{i+1}(p) \end{aligned}$$

As $p \in K^{(i)} = I_i$, this is well-defined. Furthermore, we have $D_p = z_{i+1}(p) \in \sigma^{(k)}_{q,\gamma_{i+1}}$ by definition of z_{i+1}. It remains to show that $K^{(i+1)} = I_{i+1}$. This easily follows from

$$K^{(i+1)} = \bigcup_{p \in K^{(i)}} D_p = \bigcup_{p \in I_i} z_{i+1}(p) = I_{i+1}.$$

It remains to prove that indeed $K = K^{(n)}$ (part 3.). This follows from $K^{(n)} = I_n = K$.

- $X \in \Box$

  According to Definition 25, we can write K as

$$K = \bigcup_{X \to \gamma} K^{(\gamma)} \text{ where } K^{(\gamma)} = \left\{ \bigcup_{p_1 \in z_1^{(\gamma)}(p_0)} \bigcup_{p_2 \in z_2^{(\gamma)}(p_1)} \cdots \bigcup_{p_{n-1} \in z_{n-1}^{(\gamma)}(p_{n-2})} z_n^{(\gamma)}(p_{n-1}) \right\}$$

  for some $z_1^{(\gamma)}, \ldots, z_n^{(\gamma)}$ $(n = n(\gamma) = |\gamma|)$ with

$$\begin{aligned} z_i^{(\gamma)} : I_{i-1}^{(\gamma)} &\to \bigcup_{p \in Q} \sigma^{(k)}_{q,\gamma_i} \\ p_{i-1} &\mapsto C^{(\gamma)}_{p_{i-1}} \in \sigma^{(k)}_{p_{i-1},\gamma_i}. \end{aligned}$$

  For each such γ, we use the $z_i^{(\gamma)}$ to prove that $q \xrightarrow[t \le k]{\gamma}_{\mathscr{A}} K^{(\gamma)}$. Then by applying the second saturation rule, we get that $q \xrightarrow[t \le k+1]{\gamma}_{\mathscr{A}} \bigcup_{X \to_G \gamma} K^{(\gamma)}$ and the claim follows.

  To prove the existence of the paths $q \xrightarrow[t \le k]{\gamma}_{\mathscr{A}} K^{(\gamma)}$, we proceed as in the case for $X \in \bigcirc$ for each γ by defining intermediary positions $K^{(1)}, \dots, K^{(n)}$ from the $z_i^{(\gamma)}$ The construction is analogue.

□

Lemma 17
Let $\alpha \in \mathscr{S}$ and $q \in Q$

$$q \xrightarrow[t\leq k]{\alpha}_{\mathscr{A}} K_i \textit{ for } i = 1, \dots, n \Rightarrow \sigma_{q,\alpha}^{(k)} = \{K_1, \dots, K_n\}.$$

Proof. We prove this again by induction over k.

Base case $\mathbf{k = 0}$:
As in Lemma 16.

Induction step $\mathbf{k \to k+1}$: Suppose we have transition $q \xrightarrow[t\leq k+1]{\alpha}_{\mathscr{A}} K$. The case where $\alpha = a \in T$ is again trivial. The transitions $q \xrightarrow[t\leq k+1]{a}_{\mathscr{A}} K$ labeled with a terminal word always stem from a transition $q \xrightarrow{a} p$ and by definition of the alternating automaton $\mathscr{A}_0$ we have $K = \{p\}$. But due to transition $q \xrightarrow{a} p$, we also have that $\sigma_{q,a}^{(k+1)} = \{p\}$ by definition of the system of equations. This proves the claim.

Suppose now that we have transition $q \xrightarrow[t\leq k+1]{X}_{\mathscr{A}} K$ for $X = \alpha \in N$. We want to show that $K \in \sigma_{q,X}^{(k+1)}$. Let us therefore consider the path(s) that were used in combination with a saturation rule to create $q \xrightarrow[t\leq k+1]{X}_{\mathscr{A}} K$. As the number of these paths varies depending on who owns X, we make case distinction.

- $X \in \bigcirc$

  Then, transition $q \xrightarrow[t\leq k+1]{X}_{\mathscr{A}} K$ was created using the first saturation rule in combination with a path $q \xrightarrow[t\leq k]{\gamma}{}^{*}_{\mathscr{A}} K$ for some rule $X \to_G \gamma$.

  Overview: This time, we use the intermediary positions $K^{(1)}, \dots, K^{(n)}$ $(n = |\gamma|)$ of path $q \xrightarrow[t\leq k]{\gamma}{}^{*}_{\mathscr{A}} K$ to prove that $K \in \sigma_{q,X}^{(k+1)}$. We use the $K^{(i)}$ to define appropriate functions $z_1, \dots, z_n$ with

$$\begin{aligned} z_i : & I_{i-1} \to \bigcup_{p \in Q} \sigma_{p,\gamma_i}^{(k)} \\ & p_{i-1} \mapsto C_{p_{i-1}}^{(\gamma_i)} \in \sigma_{p_{i-1},\gamma_j}^{(k)}. \end{aligned} \tag{0.2}$$

  and $I_0 = \{q\}$. The functions define a clause K' s.t.

$$K' = \{ \bigcup_{p_1 \in z_1(q)} \bigcup_{p_2 \in z_2(p_1)} \cdots \bigcup_{p_{n-1} \in z_{n-1}(p_{n-2})} z_n(p_{n-1})\} \in \sigma_{q,X}^{(k+1)}.$$

  Then claim follows by showing that $K = K'$.

  Definition of $\mathbf{z_1}, \dots, \mathbf{z_n}$: Let $K^{(1)}, \dots, K^{(n)}$ be the intermediary positions of path $q \xrightarrow[t\leq k]{\gamma}{}^{*}_{\mathscr{A}} K$. Then, by definition of the intermediary positions, it holds that

  1. there exists a transition $p \xrightarrow[t\leq k]{\gamma_1}_{\mathscr{A}} K^{(1)}$

2. for $i = 1, \ldots, n-1$, we have $K^{(i+1)} = \bigcup_{p \in K^{(i)}} D_p$, s.t. there exists a transition $p \xrightarrow{\gamma_{i+1}} D_p$
3. $K = K^{(n)}$

Then, it follows by the induction hypothesis and the two first points above that

1. $K^{(1)} \in \sigma^{(k)}_{q,\gamma_1}$ and
2. for $i = 1, \ldots, n-1$, we have that $D_p \in \sigma^{(k)}_{p,\gamma_{i+1}}$ for $p \in K^{(i)}$.

Let us now inductively define the z_i for $i = 1, \ldots, n$. To ensure that we defined $z_i(p)$ for each $p \in I_{i-1}$), we require that in each step we maintain invariant $I_i = K^{(i)}$.

First, we define $z_1(q) := K^{(1)}$, which is sufficient given that $I_0 = \{q\}$. As $K^{(1)} \in \sigma^{(k)}_{q,\gamma_1}$, the function is as desired in 0.2. The invariant also holds:

$$\begin{aligned} I_1 &= \bigcup_{p \in I_0} z_1(p) \\ &= z_1(p) \\ &= K^{(1)} \end{aligned}$$

Let now $i \in \{1, \ldots, n-1\}$ s.t. z_j for $j \leq i$ have already been defined for each $p \in I_{j-1}$ and are conform to the definition 0.2. Then, we set

$$z_{i+1}(p) = D_p \text{ for } p \in I_i.$$

From the invariant, we have that $I_i = K^{(i)}$, thus this is well defined. As $D_p \in \sigma^{(k)}_{p,\gamma_i}$, the definition of z_{i+1} is as required by definition Furthermore, we have that

$$\begin{aligned} I_{i+1} &= \bigcup_{p \in I_s} z_{i+1}(p) \\ &= \bigcup_{p \in K^{(i)}} z_{i+1}(p) \\ &= \bigcup_{p \in K^{(i)}} D_p \\ &= K^{(s+1)}. \end{aligned}$$

It remains to prove that K' as defined by $z_1, \ldots, z_n$ coincides with K. This follows from the third point above: $K' = I_n = K^{(n)} \overset{3.}{=} K$.

- $A \in \square$

  In this case, transition $q \xrightarrow[t \leq k+1]{X}_{\mathscr{A}} K$ was created using the second saturation rule in combination with paths $q \xrightarrow[t \leq k]{\gamma}{}^{*}_{\mathscr{A}} K^{(\gamma)}$ for each rule $X \to_G \gamma$ with $K = \bigcup_{X \to_G \gamma} K^{(\gamma)}$.

  Thus, we have intermediary positions $K^{(1)}, \ldots, K^{(n)}$ ($n = |\gamma|$) for each such path $q \xrightarrow[t \leq k]{\gamma}{}^{*}_{\mathscr{A}} K^{(\gamma)}$. We can apply the same construction as for the case $X \in \bigcirc$ to each sequence of intermediary positions $K^{(1)}, \ldots, K^{(n)}$ and thereby define functions

$z_1^{(\gamma)}, \dots, z_n^{(\gamma)}$ ($n = n(\gamma) = |\gamma|$) with

$$\begin{aligned} z_i^{(\gamma)} : & I_{i-1}^{(\gamma)} \to \bigcup_{p \in Q} \sigma_{q,\gamma_i}^{(k)} \\ & p_{i-1} \mapsto C_{p_{i-1}}^{(\gamma)} \in \sigma_{p_{i-1},\gamma_i}^{(k)}. \end{aligned}$$

By construction, these functions define the clauses

$$K^{(\gamma)} = \Big\{ \bigcup_{p_1 \in z_1^{(\gamma)}(p_0)} \quad \bigcup_{p_2 \in z_2^{(\gamma)}(p_1)} \quad \cdots \quad \bigcup_{p_{n-1} \in z_{n-1}^{(\gamma)}(p_{n-2})} \quad z_n^{(\gamma)}(p_{n-1}) \Big\}.$$

for each γ.

Finally, we get $K = \bigcup_{X \to_G \gamma} K^{(\gamma)} \in \sigma_{q,X}^{(k+1)}$ as expected.

□

Recall the result that we just proved.

$$\sigma_{q,\alpha}^{(k)} = \{K_1, \dots, K_m\} \iff q \xrightarrow[t \le k]{\alpha}_{\mathscr{A}} K_j \text{ for } j = 1, \dots, m.$$

for $q \in Q$ a state in A^{det} and $\alpha \in \mathscr{S}$ a sentential form. The result suggest a strong relation between the formulas and the transitions. Intuitively, this makes sense from the point of view of each method. In Section 3, we explained how transforming the formulas into DNF flattened the game to just one choice for each player. The same holds for the formulas of the alternative summary algorithm. If K is a clause of $\sigma_{q,\alpha}^{(k)}$, then refuter can enforce a terminal word w in a play from α and q s.t. $q \xrightarrow{w} p \in K$. Which state of K will be reached is decided by prover. For the saturation approach, there is a similar intuition. The choices of refuter are represented by non-determinism and the ones of prover by the sets of states on the right-hand side of transitions. If we have transition $q \xrightarrow{\alpha}_{\mathscr{A}} K$, then refuter can enforce the derivation of a terminal word w from α s.t. $q \xrightarrow{w} p \in K$. But again, she can not narrow it down to a particular state of K. This decision belong to prover.

However this does not yet explain why the k-th solution in the Kleene iteration $\sigma_{q,\alpha}^{(k)}$ corresponds to the transitions $q \xrightarrow[k]{\alpha}_{\mathscr{A}} K$ with time stamp k. Recall from Section 3 that the formulas $\sigma_{q,\alpha}^{(k)}$ capture plays starting from α with at most k applications of rules in each branch of the corresponding parse tree. With the following inductive reasoning, we can show that the same holds true for transitions $q \xrightarrow[k]{\alpha}_{\mathscr{A}} K$. For $k = 0$, there are only transitions labeled by terminals. Indeed, only plays from terminals have at most 0 transitions in each branch of their parse tree. Let us now consider the transitions of shape $q \xrightarrow[k+1]{\alpha}_{\mathscr{A}} K$ and the plays π that they capture. Suppose that $\alpha = X \in N$, otherwise the claim holds trivially. Then, the transitions from above have been constructed from paths $q \xrightarrow[k]{\gamma}{}^{*}_{\mathscr{A}} K'$, if there exists a grammar rule $\alpha \to_G \gamma = \gamma_1, \dots, \gamma_n$, using one of the saturation rules. For each of the transitions $p \xrightarrow[k]{\gamma_i}{}^{*}_{\mathscr{A}} K''$ involved in these paths holds by induction that they represent plays π_i with at most k applications in each branch of their parse tree. We construct the plays π by composing the plays π_i in the paths. Thus, the parse tree of π is rooted at X and the parse trees of π_i are appended to the root. Then, the resulting parse tree of π has at most $k+1$ applications of rules in each branch (the root counts as one application). These intuitions explain the result of Theorem 5.

The theorem furthermore shows the strong resemblance of the two algorithms in terms of their mechanics. In the summary approach, the clauses of the formula $\sigma_{q,X}^{(k+1)}$ are computed from the clauses K_i of formulas $(\sigma^{(k)}_{p,\gamma_i})_{p\in Q}$ where $X \to_G \gamma_1 \dots \gamma_n$ is a grammar rule with right-hand side X. These clauses correspond to the right-hand sides of transitions $p \xrightarrow[t\leq k]{\gamma_i}_{\mathscr{A}} K_i$ that are used to form paths of shape $q \xrightarrow[t\leq k]{\gamma}{}^{*}_{\mathscr{A}} K$. These paths in return are used with one of the saturation rules to create transitions $q \xrightarrow[t\leq k+1]{X}_{\mathscr{A}} K$.

The proof of the theorem on the other hand points out that the two methods handle the composition of plays in a different manner. In the summary algorithm, we use the matching operator : to compose plays

$$\sigma^{(k)}_{q,\alpha_1} : \sigma^{(k)}_{Q,\alpha_2} \dots : \sigma^{(k)}_{Q,\alpha_n}.$$

Or in case of the set-theoretic representation, we used the functions z_i to compose plays. In the proof, we had to translate the functions into paths

$$q \xrightarrow[t\leq k]{\alpha_1}_{\mathscr{A}} P_1 \xrightarrow[t\leq k]{\alpha_2}_{\mathscr{A}} \dots \xrightarrow[t\leq k]{\alpha_n}_{\mathscr{A}} P_n$$

and vice-versa. And indeed as we already hinted in Section 8, the saturation approach uses paths to compose plays.

9.1.5. Relation to the box-based saturation algorithm

As we mentioned in the beginning of this section, the formulas of the box-based saturation algorithm can nevertheless be determined from the transitions, but only under the right circumstances. By circumstances we mean the correct way to determinize at the beginning of the saturation algorithm. There exist several deterministic automaton A^{det} that recognize the same language as A and the saturation algorithm can be applied based on any of them. One particular is the deterministic box automaton $\mathcal{A}_M$ that we introduced in section 8. Recall that the nodes are given by the boxes of A and the transitions by the composition operator i.e. there is a transition $\tau \xrightarrow{a} \rho$ if $\rho = \tau; \rho_a$ for τ, ρ boxes and ρ_a the box of a. It is not surprising that in the case where we use the box automaton $\mathcal{A}_M$ of A for the saturation algorithm, we can actually derive the formulas from the transitions and vice-versa.

However instead of conducting a nearly identical proof as for Theorem 5, we show that there is a relation between the formulas of the box-based summary algorithm and the ones of the alternative summary algorithm if we use the box automaton of A here as well. Then we can simply reuse the statement of Theorem 5 to get the desired relation.

But before we can relate the two types of formulas, we first need to define the set-theoretic representation of the box-based formulas $\sigma^{(k)}_X$. Recall that$\sigma^{(k)}_X$ was given by

$$\sigma_X = \bigotimes_{X\to\alpha_1\dots\alpha_n} \sigma_{\alpha_1}; \dots; \sigma_{\alpha_n}, \text{ where } \bigotimes = \begin{cases} \vee & \text{if } X \in \bigcirc \\ \wedge & \text{if } X \in \square \end{cases}$$

Therefore, we need to define how the operators ;, $\vee$ and $\wedge$ behave when applied to sets of sets of boxes. The definitions for the Boolean operators are exactly the same as for the formulas of the alternative summary algorithm. The composition operator behaves similarly to the matching operator :.

Theorem 6

Let $G_1, \ldots, G_n$ be formulas. Then

$$\bigodot_{i=1}^{n} G_i = \bigcup_{\substack{z_1:I_0 \to I_0;G_1 \\ \rho_0 \mapsto \rho_0;C_1}} \bigcup_{\substack{z_2:I_1 \to I_1;G_2 \\ \rho_1 \mapsto \rho_1;C_2}} \cdots \bigcup_{\substack{z_n:I_{n-1} \to I_{n-1};G_n \\ \rho_{n-1} \mapsto \rho_{n-1};C_n}} \{ \bigcup_{\rho_1 \in z_1(\rho_0)} \bigcup_{\rho_2 \in z_2(\rho_1)} \cdots \bigcup_{\rho_{n-1} \in z_{n-1}(\rho_{n-2})} z_n(\rho_{n-1})\}$$

where

- $I_0 = \{\rho_\varepsilon\}$
- $I_j = img(z_j)$, *for* $j = 1, \ldots, n$
- $C_j \in G_j$, *for* $j = 1, \ldots, n$
- $\rho_j \in I_j$, *for* $j = 1, \ldots, n$

The only difference lies in the functions z_i, which are defined slightly different. The reason is the following. In the setting of the alternative summary algorithm, suppose we had the composition $G : F_Q$ of a formula G with a family of formulas F_Q. The functions z_i were used to replace the atomic propositions p of G by an adequate member F_p of the family F_Q and to match clauses to normalize the resulting formula to DNF. In the setting of the box-based algorithm, we may have composition $G; F$ of two formulas G and F. Instead of replacing the atomic propositions by formulas, we compose the atomic propositions τ of G with F, i.e. $\tau; F$. This composition is conducted by the functions z_i, in addition to the matching of clauses to normalize the formula to DNF.

The proof of Theorem 6 is analogue to the one of Theorem 5 and is therefore skipped.

Finally, we get the following set-theoretic representations for the formulas $\sigma_X^{(k)}$.

Definition 27
Let

- $X \to_G \gamma$ *be the grammar rules with left-hand side* X,
- $n = |\gamma|$ *the size of the right-hand side of the grammar rule* $X \to_G \gamma$,
- $I_0^{(\gamma)} = \{\rho_\varepsilon\}$ *for each rule* $X \to_G \gamma$,
- $I_i^{(\gamma)} = img(z_i^{(\gamma)})$, *for* $i = 1, \ldots, n$ *are the images of the functions* $z_i^{(\gamma)}$,
- $C_{\rho_{i-1}}^{(\gamma_i)} \in \sigma_{\gamma_i}^{(k)}$, *for* $i = 1, \ldots, n$ *are the clauses of the formulas* $\sigma_{\gamma_i}^{(k)}$
- *and* $\rho_i \in I_i^{(\gamma)}$, *for* $i = 1, \ldots, n$.

If we have now that $X \in \bigcirc$, *the set theoretic representation is given by*

$$\sigma_X^{(k+1)} = \bigcup_{X \to \gamma} \bigcup_{\substack{z_1:I_0 \to I_0;\sigma_{\gamma_1}^{(k)} \\ \rho_0 \mapsto \rho_0;C_{\rho_0}^{(\gamma_1)}}} \bigcup_{\substack{z_2:I_1 \to I_1;\sigma_{\gamma_2}^{(k)} \\ \rho_1 \mapsto \rho_1;C_{\rho_1}^{(\gamma_2)}}} \cdots \bigcup_{\substack{z_n:I_{n-1} \to I_{n-1};\sigma_{\gamma_n}^{(k)} \\ \rho_{n-1} \mapsto \rho_{n-1};C_{\rho_{n-1}}^{(\gamma_n)}}} \left\{ \bigcup_{\rho_1 \in z_1^{(\gamma)}(\rho_0)} \bigcup_{\rho_2 \in z_2^{(\gamma)}(\rho_1)} \cdots \bigcup_{\rho_{n-1} \in z_{n-1}^{(\gamma)}(\rho_{n-2})} z_n^{(\gamma)}(\rho_{n-1})\right\}$$

In case $X \in \square$, *we have*

$$\sigma_X^{(k+1)} = \Bigg\{ \bigcup_{X \to_G \gamma} K^{(\gamma)} \mid$$

$$K^{(\gamma)} \in \bigcup_{\substack{z_1^{(\gamma)}:I_0^{(\gamma)} \to I_0;\sigma_{\gamma_1}^{(k)} \\ \rho_0 \mapsto \rho_0;C_{\rho_0}^{(\gamma_1)}}} \bigcup_{\substack{z_2^{(\gamma)}:I_1^{(\gamma)} \to I_1;\sigma_{\gamma_2}^{(k)} \\ \rho_1 \mapsto \rho_1;C_{\rho_1}^{(\gamma_2)}}} \cdots \bigcup_{\substack{z_n^{(\gamma)}:I_{n-1}^{(\gamma)} \to I_{n-1};\sigma_{\gamma_n}^{(k)} \\ \rho_{n-1} \mapsto \rho_{n-1};C_{\rho_{n-1}}^{(\gamma_n)}}}$$

$$\Bigg\{ \bigcup_{\rho_1 \in z_1^{(\gamma)}(\rho_0)} \bigcup_{\rho_2 \in z_2^{(\gamma)}(\rho_1)} \cdots \bigcup_{\rho_{n-1} \in \rho_{n-1}^{(\gamma)}(\rho_{n-2})} z_n^{(\gamma)}(\rho_{n-1}) \Bigg\} \Bigg\}.$$

Relation of the two types of formulas Suppose now that we used the box automaton $\mathcal{A}_M$ as base to compute the formulas $\sigma_{q,X}^{(k)}$. This means that the states q in the formulas correspond boxes of A i.e. $q = \rho$ for some box $\rho \in B(A)$. Thus, both types of formulas $\sigma_X^{(k)}$ and $\sigma_{\rho,X}^{(k)}$ belong to the same domain, namely the positive Boolean formulas over the set of boxes of A. This allows us to apply the matching operator : with a formula $\sigma_X^{(k)}$ on the left side, i.e. $\sigma_X^{(k)} : F_{B(A)}$ for a family of formulas $F_{B(A)}$ in the following proof.

Theorem 7
Let ρ be a state of the box automaton $\mathcal{A}_M$ of A, i.e. ρ is a box of A and X a non-terminal. Then, the following holds for all $k \in \mathbb{N}_0$.

$$\sigma_{\rho,X}^{(k)} = \rho; \sigma_X^{(k)}$$

Proof. We prove the claim by induction over k.

Base case: $k = 0$

In this case, the proof holds trivially, because $\sigma_{\rho,X}^{(0)} = \sigma_X^{(0)} = \mathit{false}$ and then also $\rho; \sigma_X^{(0)} = \mathit{false}$.

Induction step: $k \to k+1$

Then, $\sigma_{\rho,X}^{(k)}$ is of the following form.

$$\sigma_{\rho,X}^{(k+1)} = \bigotimes_{X \to \alpha_1 \dots \alpha_n} \sigma_{\rho,\alpha_1}^{(k)} : \sigma_{B(A),\alpha_2}^{(k)} \cdots : \sigma_{B(A),\alpha_n}^{(k)}, \text{ where } \bigotimes = \begin{cases} \vee & \text{if } X \in \bigcirc \\ \wedge & \text{if } X \in \Box \end{cases}$$

Recall that the set of boxes $B(A)$ of A coincides with the set of states of $\mathcal{A}_M$.

We can apply the induction hypothesis to $\sigma_{\rho,\alpha_1}^{(k)}$ and get

$$\begin{aligned} \sigma_{\rho,X}^{(k+1)} &= \bigotimes_{X \to \alpha_1 \dots \alpha_n} \rho; \sigma_{\alpha_1}^{(k)} : \sigma_{B(A),\alpha_2}^{(k)} \cdots : \sigma_{B(A),\alpha_n}^{(k)} \\ &= \rho; \left[\bigotimes_{X \to \alpha_1 \dots \alpha_n} \sigma_{\alpha_1}^{(k)} : \sigma_{B(A),\alpha_2}^{(k)} \cdots : \sigma_{B(A),\alpha_n}^{(k)} \right] \end{aligned}$$

by definition of the composition operator ;.

To prove the theorem, it remains to show that

$$G : \sigma_{B(A),\alpha_i}^{(k)} = G; \sigma_{\alpha_i}^{(k)} \text{ for } i = 2, \dots, n.$$

We prove this claim by a structural induction over G.

Base case $\mathbf{G} = \tau \in \mathbf{B(A)}$:

$$\begin{aligned}G : \sigma^{(k)}_{B(A),\alpha_i} &= \tau : \sigma^{(k)}_{B(A),\alpha_i}\\ &= \sigma^{(k)}_{\tau,\alpha_i}\\ &= \tau;\sigma^{(k)}_{\alpha_i} = G;\sigma^{(k)}_{\alpha_i}\end{aligned}$$

The last step follows from the induction hypothesis on k.

Induction step $\mathbf{G = G_1 \bigotimes G_2}$:

We get from the definitions of : and ; that

$$\begin{aligned}G : \sigma^{(k)}_{B(A),\alpha_i} &= (G_1 \bigotimes G_2) : \sigma^{(k)}_{B(A),\alpha_i}\\ &= G_1 : \sigma^{(k)}_{B(A),\alpha_i} \bigotimes G_2 : \sigma^{(k)}_{B(A),\alpha_i}\\ &\overset{IH}{=} G_1;\sigma^{(k)}_{\alpha_i} \bigotimes G_2;\sigma^{(k)}_{\alpha_i}\\ &= (G_1 \bigotimes G_2);\sigma^{(k)}_{\alpha_i} = G;\sigma^{(k)}_{\alpha_i}.\end{aligned}$$

□

Intuitively, the result can be explained as follows. Suppose terminal word w can be derived from X. Then formula $\sigma_{\rho,X}$ captures the target state τ of the run of w in $\mathcal{A}_M$ starting from ρ. But as the transitions in ρ_A correspond to the composition operator, this means that $\tau = \rho;\rho_w$ On the other hand, σ_X captures the box ρ_w of w. The structure of both formulas is the same as they both consider the words derivable from X and identically handle the choices of the players. They only differ in the atomic proposition. But it suffices to compose box ρ in front of every atomic proposition in σ_X to get the atomic proposition in $\sigma_{\rho,X}$. Using the computation rules for the composition operator, we can pull the box ρ to the front of σ_X and the claim follows.

Comparison: Box-based summarization vs. Saturation By combining the results of Theorem 5 and Theorem 7, we can deduce the relation between the box-based summary algorithm and the saturation approach based on the deterministic box automaton $\mathcal{A}_M$.

Theorem 8
Let $\rho \in B(A)$ be a box of A and $\alpha \in \mathscr{S}$. Then we have

$$\rho;\sigma^{(k)}_{\alpha} = \{K_1, \ldots, K_m\} \iff \rho \xrightarrow[t\leq k]{\alpha}_{\mathscr{A}} K_j \text{ for } j = 1, \ldots, m.$$

Although we again have a strong relation between the clauses of the formulas and the transitions of the saturated automaton, this result clearly suggests that the saturation method is not able to make use of the power of the boxes. Note therefore that the clauses of formula $\rho;\sigma^{(k)}_{\alpha}$ are of shape $\rho;K'$ for $K' \in \sigma^{(k)}_{\alpha}$.

Suppose we want to compose the plays starting from α_1 with the plays starting from α_2 ($\alpha_i \in \mathscr{S}$). In the summary approach, the boxes allowed us to compute the plays starting from α_1 resp. α_2 independently in form of formulas σ_{α_1} resp. σ_{α_2} and compose them afterwards with the ;-operator: $\sigma_{\alpha_1};\sigma_{\alpha_2}$. In terms of the box automaton $\mathcal{A}_M$, independently means that we capture the target states of runs starting from ρ_ε on words derivable from α_i.

In the saturation approach however, we compose plays using paths in the alternating automaton. In this case, we are interested in the paths of shape $\rho_\varepsilon \xrightarrow{\alpha_1\alpha_2}{}^{*}_{\mathscr{A}} P \subseteq B(A)$. For the paths, we need all possible transitions $\rho_\varepsilon \xrightarrow{\alpha_1}_{\mathscr{A}} P_1$, and $\tau \xrightarrow{\alpha_2}_{\mathscr{A}} P_2$ for each $\tau \in P_1$. Theorem 8 shows that for $\tau, \tau' \in P_1$, the latter transitions are of shape $\tau \xrightarrow{\alpha_2}_{\mathscr{A}} \tau;K_i$ resp. $\tau' \xrightarrow{\alpha_2}_{\mathscr{A}} \tau';K_i$ for each $K_i \in \sigma_{\alpha_2}$. Thus, the saturation approach computes the same information several times, more precisely it does not compute the information about the plays from α_2 independently of the plays from α_1. The reason is that the composition by paths (see Section 8) does not allow this independent computation and thus the saturation approach can not make use of the power of the boxes.

For this reason and because the box automaton in general is not the smallest deterministic automaton recognizing $\mathcal{L}(A)$ (see for example Figure 35), it is not advisable to use the saturation algorithm in combination with the box automaton. The saturation approach benefits from a small deterministic automaton, thus it is preferable to use a minimal deterministic automaton recognizing $\mathcal{L}(A)$.

This concludes the comparison of the summary approach with the saturation approach. We now consider the comparison with the Guess & Check method.

9.2. Summarization approach vs. Guess & Check approach

In this section, we intend to compare the nodes contained in the k-th fixed-point approximant $\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ with the formulas $\sigma^{(k)}_X$. More precisely, we want to compare the verify-nodes with the formulas. Intuitively, refuter wins from a verify-node $\mathit{Verify}(X, P, q)$ if she is able to enforce the derivation of a terminal word w from X s.t. $q \xrightarrow{w} p \in P$. In Section 3 on the other hand, we argued that formula σ_X contains clause K if refuter can enforce the derivation of a terminal word w s.t. the box ρ_w of w is contained in K. It would thus not be surprising if there was a relationship between the clauses K and the predictions P in the verify-nodes.

At this point, the reader may ask whether we could also use the claim-nodes, where the predictions are made, for the comparison. The answer is negative. This is related to the fact that in the Guess & Check method the tasks of computing the possible plays and the task of deciding whether refuter can enforce a winning one are not separated. Refuter only wins at a claim-node $\mathit{Claim}(X\gamma, P, P', q)$ if two conditions hold true. On one hand, refuter must be able to derive a terminal w from X s.t. $q \xrightarrow{w} p$. This is the property that we are interested in. However, she must also be able to derive a terminal word w' from γ s.t. $p \xrightarrow{w'} p' \in P'$ for every $p \in P$. Thus, the claim-nodes in the attractor will not match with the formulas.

Let us thus try to determine a relationship between the formulas $\sigma^{(k)}_X$ and the verify-nodes in the antichain attractor set $\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$. But as for the comparison with the saturation approach, deriving the formulas from the verify-nodes does not seem possible. Figure 36 shows an example of a context-free game. On the left, the formula which is computed by the summary approach is depicted. On the right, we have the verify-nodes that are contained in the antichain attractor. Let us try to derive the formulas from the verify-nodes. We argued that the predictions should correspond to clauses. Thus, as we have two predictions, the formula should be of shape $\tau \vee \rho$. As the verify-nodes $\mathit{Verify}(X, \{q_0\}, q_0)$ and $\mathit{Verify}(X, \{q_1\}, q_0)$ are contained in the attractor, refuter can enforce the derivation of a word w from X s.t. $q_0 \xrightarrow{w} q_0$ or $q_0 \xrightarrow{w} q_1$. At this point, we would like to assemble the clauses τ and ρ from these state changes. However, we only have state changes with q_0 as left-hand side as we only have verify-nodes with state q_0 in the attractor. But for the boxes, we also need the state changes from q_1 on.

Note that even if the attractor contained verify-nodes with state q_1, we would still not be able to derive the formulas. As in the previous section, we would not know how to pair the verify-nodes with states q_0 resp. q_1 to assemble the boxes. Thus, in any case we are not able to derive the formulas from the verify-nodes that are contained in the attractor.

Due to the fact that the attractor may contain only verify-nodes $\mathit{Verify}(X, P, q)$ for some of the states $q \in Q$, it makes sense to compare them to the formulas of the alternative summary algorithm. If the attractor contains verify-nodes $\mathit{Verify}(X, P, q)$ for some state q, we should be able to establish a relation to the formula $\sigma_{q,X}$.

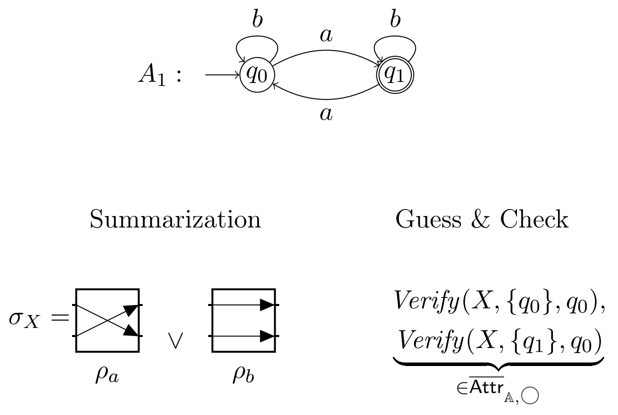
The rest of this section is organized as follows. We first show that there is indeed a relation between the verify-nodes $\mathit{Verify}(X, P, q)$ in the k-th fixed-point approximant $\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ and the formulas $\sigma^{(k)}_{q,X}$. Then, we show that the formulas $\sigma^{(k)}_X$ are related to the verify-nodes of the k-th fixed-point approximant of the attractor if we use the box automaton as base for the construction of the game graph.

9.2.1. Comparison: Summarization vs. Guess & Check

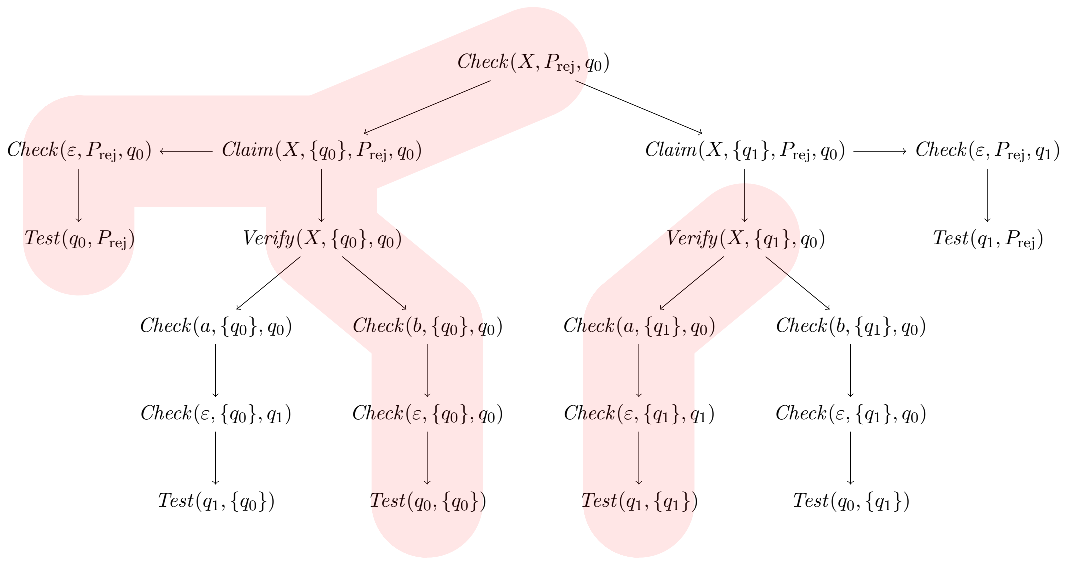
As we will show, we can compute the formulas $\sigma_{q,X}$ directly from the attractor $\mathsf{Attr}_{\mathbb{A},\bigcirc}$. It is even possible to determine the intermediary solutions $\sigma^{(k)}_{q,X}$ from the intermediary attractor sets $\mathsf{Attr}^{(i)}_{\mathbb{A},\bigcirc}$. However, we have to slightly modify the definition of the attractor.

Instead of advancing step-by-step in the game graph $\mathcal{G}_{\mathrm{GC}}$ by only including appropriate direct predecessors, we want to iterate the attractor construction until we reach a verify-node on the path (or can not progress any further). In other words, in each attractor step we

$G: \qquad X_\square \to a \mid b$


A_1 :
q_0
q_1
a
a
b
b
Summarization
Guess & Check
$\sigma_X =$
$\vee$
ρ_a
ρ_b
$\underbrace{\mathit{Verify}(X,\{q_0\},q_0),\ \mathit{Verify}(X,\{q_1\},q_0)}_{\in\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}}$

(a) Only verify-nodes with state q_0 appear in the game graph.


$\mathit{Check}(X, P_{\mathrm{rej}}, q_0)$
$\mathit{Check}(\varepsilon, P_{\mathrm{rej}}, q_0)$
$\mathit{Claim}(X, \{q_0\}, P_{\mathrm{rej}}, q_0)$
$\mathit{Claim}(X, \{q_1\}, P_{\mathrm{rej}}, q_0)$
$\mathit{Check}(\varepsilon, P_{\mathrm{rej}}, q_1)$
$\mathit{Test}(q_0, P_{\mathrm{rej}})$
$\mathit{Verify}(X, \{q_0\}, q_0)$
$\mathit{Verify}(X, \{q_1\}, q_0)$
$\mathit{Test}(q_1, P_{\mathrm{rej}})$
$\mathit{Check}(a, \{q_0\}, q_0)$
$\mathit{Check}(b, \{q_0\}, q_0)$
$\mathit{Check}(a, \{q_1\}, q_0)$
$\mathit{Check}(b, \{q_1\}, q_0)$
$\mathit{Check}(\varepsilon, \{q_0\}, q_1)$
$\mathit{Check}(\varepsilon, \{q_0\}, q_0)$
$\mathit{Check}(\varepsilon, \{q_1\}, q_1)$
$\mathit{Check}(\varepsilon, \{q_1\}, q_0)$
$\mathit{Test}(q_1, \{q_0\})$
$\mathit{Test}(q_0, \{q_0\})$
$\mathit{Test}(q_1, \{q_1\})$
$\mathit{Test}(q_0, \{q_1\})$

(b) Part of the game graph $\mathcal{G}_{\mathrm{GC}}$ for the context-free game defined in Figure 39a. The nodes that are marked in red are all nodes of the game graph $\mathcal{G}_{\mathrm{GC}}$ that are contained in the attractor $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}$.

Figure 39.: Reconstruction of the formulas fails.

ascend as far in $\mathcal{G}_{\mathrm{GC}}$ as possible and only stop if we reach a verify-node on the path or no further nodes can be included (Figure 40). The verify-nodes can be understood as barriers for the attractor construction.

Formally, we can define the modified attractor $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}$ as follows.

Definition 28

$$\overline{\mathsf{Attr}}^{(0)}_{\mathbb{A},\bigcirc} = \{\mathit{Test}(q,P) \mid q \in P\}$$

$$\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc} = \mathbb{A}\Big(\mathit{IterateAttr}(\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}, 1)\Big)$$

$$\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc} = \bigcup_{k\geq 0} \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$$

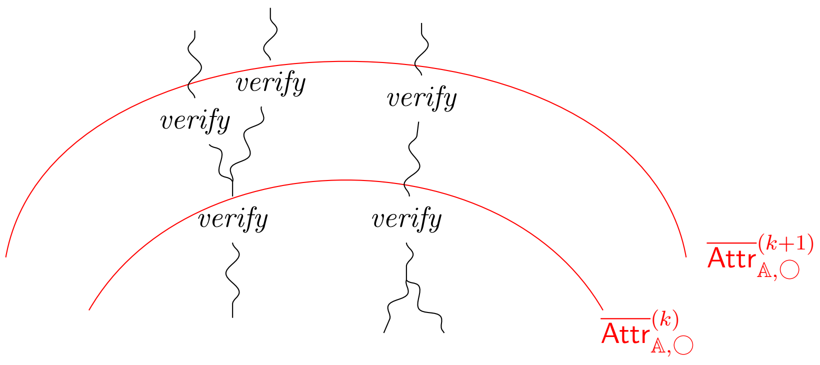
$$\textit{where IterateAttr}(M, i) = \begin{cases} M_{new} \cup \textit{IterateAttr}(M_{new}, i_{new}) \cup M_{verify} & \textit{if } i = 1 \\ \emptyset & \textit{if } i = 0 \end{cases}$$

with

$$\begin{aligned} M_{new} = & \{v \in \mathcal{V}, v \textit{ not a verify-node} \mid v \in \bigcirc \textit{ and } \exists\ (v, v') \in \mathcal{E} \textit{ with } v' \in M\} \\ & \cup \{v \in \mathcal{V}, v \textit{ not a verify-node} \mid v \in \square \textit{ and } \forall\ (v, v') \in \mathcal{E} \textit{ have } v' \in M\} \\ & \cup M \end{aligned}$$

$$i_{new} = \begin{cases} 1 & \textit{if } M \subsetneq M_{new} \\ 0 & \textit{if } M = M_{new} \end{cases}$$

$$\begin{aligned} M_{verify} = & \{v \in \mathcal{V}, v \textit{ verify-node} \mid v \in \bigcirc \textit{ and } \exists\ (v, v') \in \mathcal{E} \textit{ with } v' \in M\} \\ & \cup \{v \in \mathcal{V}, v \textit{ verify-node} \mid v \in \square, v \textit{ and } \forall\ (v, v') \in \mathcal{E} \textit{ have } v' \in M\} \end{aligned}$$



verify
verify
verify
verify
verify
$\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$
$\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$

Figure 40.: Intuition behind the modified attractor.

The idea behind the construction is the following. Suppose we want to compute $\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. The function *IterateAttr* is used to iterate a regular one-step attractor construction. Initially, the input of *IterateAttr* is the set $\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$. Then, in each step sets M_{new} and M_{verify} are computed from the input M ($M \subseteq \mathcal{V}$). Set M_{new} contains all non-verify-nodes that would be contained in the regular one-step attractor (where only direct predecessors are considered) when applied to set M. Set M_{verify} on the other hand contains all verify-nodes of the one-step attractor of M. We need to make this separation to ensure that only the nodes in M_{new} are used for further attractor computations by *IterateAttr*. The nodes in M_{verify} are simply added to the fixed-point approximant $\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. The input i_{new} serves as flag to ensure the termination of the attractor construction. If no new nodes are collected in M_{new}, the flag is set to 0 and the computation terminates. Note that the computation always terminates as the set of nodes $\mathcal{V}$ in the game graph $\mathcal{G}_{\text{GC}}$ is finite. At the end, we apply the antichain operator $\mathbb{A}$ to $\textit{IterateAttr}(\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}, 1)$ to get rid of the non-$\preceq$-minimal elements.

Although the definition of the alternative attractor $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}$ differs a bit from the definition of the antichain attractor $\mathsf{Attr}_{\mathbb{A},\bigcirc}$ from Section 5, the new attractor $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}$ also allows us to determine the winner of the *GC game*. In fact, Theorem 2 can be proven analogously for $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}$.

The rough idea behind this new attractor is to count the number of verify-nodes on paths in $\mathcal{G}_{\text{GC}}$. As grammar rules in the $\mathcal{G}_{\text{GC}}$ are always applied after the verify-nodes, counting the number of verify-nodes comes down to counting the number of rules that have been applied. Recall from Subsection 9.1 that the formulas $\sigma^{(k)}_{q,X}$ captured plays for which at most k rules were applied in each branch of the parse tree. Thus, it makes sense to limit the number of applied rules on the paths in $\mathcal{G}_{\mathrm{GC}}$ when we aim to compute the intermediary solutions.

Before we present the theorem describing the relation between the solutions $\sigma_{q,X}^{(k)}$ and the fixed-point approximant $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$, we state a few simple properties of $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$, which will be useful in the proof of the theorem. The statements allow us to conclude that particular node(s) are included in the attractor from the inclusion of its pre- or successor(s) (statements 1,2). As the verify-nodes are handled differently in the attractor $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}$ than the other nodes, we need special cases if any of the involved nodes is a verify-node (statements 3,4).

Lemma 18
Let $v \in \mathcal{V}$ be a node in the game graph $\mathcal{G}_{GC}$.

1. *Let $v \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$. Then, it holds that*
   - *If $v \in \mathcal{V}_{\bigcirc}$, then $\exists$ transition $(v, v') \in \mathcal{E}$ s.t. $v' \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$.*
   - *If $v \in \mathcal{V}_{\square}$, then $\forall$ transitions $(v, v') \in \mathcal{E}$ holds $v' \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$.*
2. *Let $v'_1, \dots, v'_m$ be the successors of v and assume none of the nodes $v, v'_1, \dots, v'_m$ is a verify-node.*
   - *If $v \in \mathcal{V}_{\bigcirc}$ and $v'_i \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$ for some $i \in \{1, \dots, n\}$, then $v \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$.*
   - *If $v \in \mathcal{V}_{\square}$ and $v'_i \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$ for all $i \in \{1, \dots, n\}$, then $v \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$.*
3. *Let v be a claim-node and v'_0 be the succeeding verify-node resp. $v'_1, \dots, v'_n$ the succeeding check-nodes. Then it holds that*
$$v \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)} \iff v'_0 \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k-1)} \text{ and } v'_i \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)} \text{ for } i = 1, \dots, n.$$
4. *Let v be a verify-node.*
   - *If $v \in \mathcal{V}_{\bigcirc}$ and $v'_i \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$ for some $i \in \{1, \dots, n\}$, then $v \in IterateAttr(\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}, 1)$.*
   - *If $v \in \mathcal{V}_{\square}$ and $v'_i \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$ for all $i \in \{1, \dots, n\}$, then $v \in IterateAttr(\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}, 1)$.*

Proof. For the proof, we use the following notation. In the attractor set $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$, sets M_{new} resp. M_{verify} are computed iteratively by function *IterateAttr*. If a set was computed in the i-th iteration, we denote it by $M_{\text{new}}^{(i)}$ resp. $M_{\text{verify}}^{(i)}$ for $i = 1, \dots, n$, where n is the iteration where i_{new} is set to 0. For the attractor set $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k-1)}$, we denote the corresponding sets by $M'^{(i)}_{\text{new}}$ resp. $M'^{(i)}_{\text{verify}}$ for $i = 1, \dots, n'$.

For the statements one, two and four, we only prove the part for refuter. The case for prover is analogue.

Claim 1:

We prove the claim by contraposition. Assume that no successor of v is contained in $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$. Thus none of the successors is contained in any of the sets $M_{\text{new}}^{(i)}$ or $M_{\text{verify}}^{(i)}$, $i \leq n$. But from the definitions of the sets M_{new} and M_{verify}, it is clear that a node is only included in iteration j if at least one of its successors is included in $M_{\text{new}}^{(j-1)}$. Thus, $v \notin M_{\text{new}}^{(i)}$ for any $i \leq n$ and thus $v \notin \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$.

Claim 2:

Let v' be a successor of v s.t. $v' \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$. As v' is no verify-node, $v' \in M_{\text{new}}^{(i)}$ for some $i \leq n$. But then by definition of M_{new} resp. M_{verify}, either $v \in M_{\text{new}}^{(i+1)}$ or $v \in M_{\text{verify}}^{(i+1)}$, depending on the type of node of v. In any case, $v \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$ as v is also not a verify-node (meaning that it will not be discarded by the $\mathbb{A}$-operator).

Claim 3:

For the direction from left to right assume that either $v'_0 \notin \overline{\mathsf{Attr}}^{(k-1)}_{\mathbb{A},\bigcirc}$ or $v'_j \notin \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ for some $j \in \{1, \dots, m\}$. In the first case, we may have that $v'_0 \in M^{(i)}_{\text{verify}}$ for some $i \leq n$, but as the verify-nodes are treated separately v'_0 will never appear in any of the sets $M^{(i)}_{\text{new}}$ for any $i \leq n$. In the second case, we get that $v'_j \notin M^{(i)}_{\text{new}}$ for any $i \leq n$ (as in Claim 1). Thus, it follows that also $v \notin M^{(i)}_{\text{new}}$ for any $i \leq n$ and thereby $v \notin \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$.

For the other direction, assume that $v'_0 \in \overline{\mathsf{Attr}}^{(k-1)}_{\mathbb{A},\bigcirc}$ and $v'_j \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ for $j = 1, \dots, m$. This means that $v'_0 \in M^{(0)}_{\text{new}}$ for any $i \leq n$ and that for each of the v'_j holds that they are included in some $M^{(i_j)}_{\text{new}}$. If we take take the maximum $i_{\max}$ of all the $i_1, \dots, i_m$, we get that $v'_0, v'_1, \dots, v'_m \in M^{(i_{\max})}_{\text{new}}$ as we have $M^{(i)}_{\text{new}} \subseteq M^{(i+1)}_{\text{new}}$ for $0 \leq i \leq n-1$. But then, $v \in M^{(i_{\max}+1)}_{\text{new}}$ by definition of M_{new} and thereby $v \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$.

Claim 4:

Let v' be a successor of v s.t. $v' \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$. As v' is no verify-node (the successors of verify-nodes are never verify-nodes themselves), $v' \in M^{(i)}_{\text{new}}$ for some $i \leq n$. But then by definition of M_{new} resp. M_{verify}, either $v \in M^{(i+1)}_{\text{new}}$ or $v \in M^{(i+1)}_{\text{verify}}$, depending on the type of node of v. In any case, $v \in \mathit{IterateAttr}(\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}, 1)$. We can not conclude that $v \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ as v may be discarded by the $\mathbb{A}$-operator.

□

With the helping lemma at hand, we can prove the following theorem, stating the relation between the formulas $\sigma^{(k)}_{q,X}$ and the verify-nodes $\mathit{Verify}(X, P, q)$ in the attractor set $\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$.

Theorem 9
Let $X \in N$.

1. *If $\mathit{Verify}(X, K, q) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ then $K \in \sigma^{(k)}_{q,X}$ (reduced).*
2. *If $K \in \sigma^{(k)}_{q,X}$ (reduced) then $\mathit{Verify}(X, K, q) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ given that $\mathit{Verify}(X, K, q) \in \mathcal{V}$.*

We call a CNF-formula $\sigma^{(k)}_{q,X}$ reduced if it does not contain any two clauses K, K' s.t. $K \subseteq K'$. This restriction is sound as a non-restricted CNF-formula is always logically equivalent to the corresponding restricted formula. We have to presume that the formula is restricted because we only consider inclusion minimal predictions P of verify-nodes $\mathit{Verify}(X, P, q)$ in the antichain attractor.

For the second part of the theorem, we have to presume that the verify-node is included in the game graph $\mathcal{G}_{\text{GC}}$, as there may be states q s.t. no verify-node $\mathit{Verify}(X, P, q)$ are included in $\mathcal{G}_{\text{GC}}$ for a fixed non-terminal X. In Figure 39 for example, $\mathit{Verify}(X, P, q_1) \notin \mathcal{V}$ for any prediction P. There may even be a non-terminal X s.t. no verify-node $\mathit{Verify}(X, P, q)$ is contained in the game graph at all. Both of these properties are due to the fact that the game graph $\mathcal{G}_{\text{GC}}$ is constructed in a top-down fashion. Intuitively, a run in the $\mathcal{G}_{\text{GC}}$ from the starting node corresponds to a derivation process starting from S. If the run ends in a node $\mathit{Verify}(X, P, q)$, the derivation process should end at a sentential form $wX\alpha$ s.t. $q_0 \xrightarrow{w} q$ in the deterministic automaton A^{det}. Thus, if no (appropriate) such derivation process exists, no verify-node $\mathit{Verify}(X, P, q)$ with state q will be in $\mathcal{V}$. Unfortunately, as the predictions in the $\mathcal{G}_{\text{GC}}$ can be arbitrary sets, we can not make such a strong statement. In fact, as Figure 41a shows, the game graph may include verify-nodes even if no appropriate derivation process exists. However, due to the fact that we do not have predictions for terminal words, at least some verify-nodes with no appropriate derivation process can be excluded (Figure 41b).

Even though the predictions in the nodes are arbitrary, the sentential forms stored in the nodes are not. The sentential forms appearing in the nodes are only based on the grammar rules. If there is no derivation process from S reaching X at all, no such verify-node $\mathit{Verify}(X, P, q)$ will be contained in $\mathcal{G}_{\text{GC}}$ at all.

$$\begin{tikzcd}
 & & \mathit{Verify}(S,P,q) \arrow[d] & & \\
 & & \mathit{Check}(YZ,P,q) \arrow[dl] \arrow[dr] & & \\
 & \mathit{Claim}(YZ,P_{\text{arbitr}},q) \arrow[dl] \arrow[dr] & & \mathit{Claim}(\dots) & \\
\mathit{Verify}(\dots) & & \mathit{Check}(Z,P,q_{\text{arbitr}}) \arrow[d] & & \\
 & & \mathit{Claim}(Z,P',P,q_{\text{arbitr}}) \arrow[dl] \arrow[dr] & & \\
 & \mathit{Verify}(Z,P',q_{\text{arbitr}}) & & \mathit{Check}(\dots) &
\end{tikzcd}$$

(a) We have $\mathit{Verify}(Z,P',q_{\text{arbitr}}) \in \mathcal{V}$ for any state q_{arbitr}.

$$\begin{tikzcd}
 & & \mathit{Verify}(S,P,q) \arrow[d] & & \\
 & & \mathit{Check}(YaZ,P,q) \arrow[dl] \arrow[dr] & & \\
 & \mathit{Claim}(YaZ,P_{\text{arbitr}},q) \arrow[dl] \arrow[dr] & & \mathit{Claim}(\dots) & \\
\mathit{Verify}(\dots) & & \mathit{Check}(aZ,P,q_{\text{arbitr}}) \arrow[d] & & \\
 & & \mathit{Check}(Z,P,q' \neq q) \arrow[d] & & \\
 & & \mathit{Claim}(Z,P',P,q') \arrow[dl] \arrow[dr] & & \\
 & \mathit{Verify}(Z,P',q') & & \mathit{Check}(\dots) &
\end{tikzcd}$$

(b) Suppose that state q has no incoming edges labeled by a. Then $\mathit{Verify}(Z,P',q) \notin \mathcal{V}$.

Figure 41.

Proof idea: Suppose we have $K \in \sigma_{q,X}^{(k)}$ and want to show that node $\mathit{Verify}(X,K,q) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$ (given that it is contained in the game graph at all). From the set-theoretic representation of $\sigma_{q,X}^{(k)}$, we know that there exist appropriate functions $z_1,\dots,z_n$ defining clause K. On the other hand, we know that $\mathit{Verify}(X,K,q) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$ if at least one (in case $X \in \bigcirc$) or all (if $X \in \Box$) of its successors $\mathit{Check}(\alpha,K,q)$ are contained in $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$. We will use the functions $z_1,\dots,z_n$ to define a winning strategy from the check-node(s) $\mathit{Check}(\alpha,K,q)$. More precisely, the functions provide the correct predictions for prover in the claim-nodes s.t. she wins in both the verify- and in any of the skip-branches. If there exists such a winning strategy, $\mathit{Check}(\alpha,K,q)$ must be contained in the attractor.

For the other direction, we the winning strategy to define the functions $z_1,\dots,z_n$ by taking the predictions made during the winning plays as function values of the z_i.

We now prove each direction of the theorem separately. For the first direction, we drop the property that the formulas have to be restricted at the moment. It will be reestablished later.

Lemma 19
Let $X \in N$.
$v = \mathit{Verify}(X,K,q) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)} \Rightarrow K \in \sigma_{q,X}^{(k)}$.

Proof. We prove this by induction over k.

Base case $\mathbf{k = 0}$:

If $k = 0$, then $\sigma_{q,X}^{(k)} = \{\}$ by definition. Thus $\sigma_{q,X}^{(k)}$ does not contain any clause. By definition, $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(0)}$ does not contain any verify-node. The claim follows.

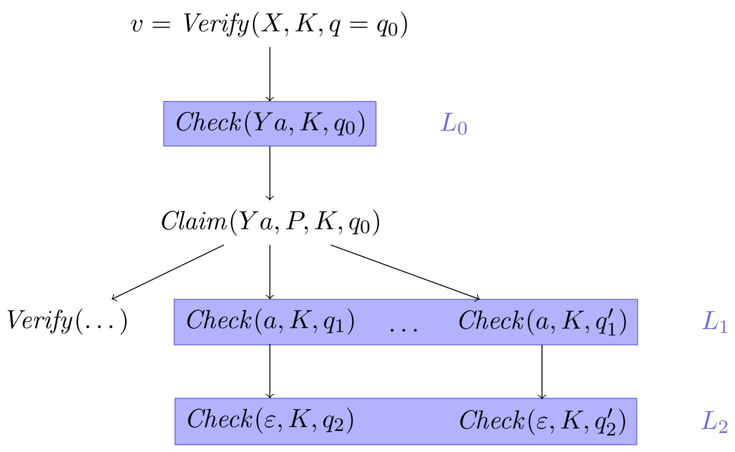


Figure 42.: Example of layers L_0, L_1, L_2 for starting node $v = Verify(X, K, q)$. All depicted nodes are contained in the attractor $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}$.

Induction step $\mathbf{k \to k+1}$: Let $v = Verify(A, K, q) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. Depending on the owner of X, either at least one or all successors of v have to be contained in $\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ (Lemma 18 (1.)). Thus, we distinguish two cases:

1. $\boldsymbol{X \in \bigcirc}$

   Then, there exists at least one successor node $Check(\gamma, K, q) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ with $X \to_G \gamma$ a grammar rule. The goal is to prove that $K \in \sigma^{(k+1)}_{q,X}$.

   To prove the claim, we define $z_1, \dots, z_n$ ($n = |\gamma|$) s.t.

$$\begin{aligned} z_i :& I_{i-1} \to \bigcup_{p \in Q} \sigma^{(k)}_{p,\gamma_i} \\ & p \mapsto C^{(\gamma_i)}_p \in \sigma^{(k)}_{p,\gamma_i} \end{aligned} \tag{0.3}$$

   and thereby construct clause

$$K' = \{ \bigcup_{q_1 \in z_1(q)} \quad \bigcup_{q_2 \in z_2(q_1)} \quad \dots \quad \bigcup_{q_{n-1} \in z_{n-1}(q_{n-2})} z_n(q_{n-1}) \} \in \sigma^{(k+1)}_{q,X}$$

   where $I_0 = \{q\}$ and $I_i = img(z_i)$ for $i = 1, \dots, n$.

   Finally, we show that $K' = K$, which proves the claim.

   To define these z_i, we use the concept of layers L_i for $i = 0, \dots, n$ in the game graph $\mathcal{G}_{\mathrm{GC}}$. We already used layers in Section 5, but we briefly recall the concept. Layers L_{i-1} are sets of check nodes of shape $Check(\gamma_{[i,n]}, K, q_{i-1})$, which are collected by moving downwards inside the attractor $\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ starting at $v = Verify(X, K, q)$ in $\mathcal{G}_{\mathrm{GC}}$. Layer L_{i+1} is obtained from layer L_i by collecting all next occurrences of check-nodes that are reachable from a check-node in L_i inside of the attractor $\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. The process is depicted in Figure 42.

   We are interested in the states of the check-nodes in the layers, therefore we use the following definition.

$$state(L_i) := \{q \mid Check(\gamma_{[i+1,n]}, P, q) \in L_i\}$$

   As we will see, the sets $state(L_i)$ of the layers L_i correspond to the images I_i of the functions z_i. To compute the actual function values $z_i(p)$ for $p \in I_{i-1}$, we consider the successors in L_i of the check-node $u = \mathit{Check}(\gamma_{[i,n]}, K, q) \in L_{i-1}$. If check-nodes $\mathit{Check}(\gamma_{[i+1,n]}, K, q_j) \in L_i$ for $j = 1, \dots, m$ are reachable from u, then $z_i(p) = \{q_1, \dots, q_m\}$. In the example of Figure 42, we would have that $z_1(q_0) = \{q_1, \dots, q_1'\}$ and $z_2(q_1) = \{q_2\}$.

We now make this construction precise and show that the functions $z_1, \dots, z_n$ are well-defined and as desired in Equation 0.3.

Layer construction We inductively define the $L_0, \dots, L_n$ and thereby also the $z_1, \dots, z_n$. In order to guarantee that the z_i are well-defined and as we claimed in Equation 0.3, we maintain the following invariants:

a) $\mathit{state}(L_i) = I_i$

b) $L_i \subseteq \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ for $i = 0, \dots, n$.

We construct the layers inductively as follows.

$\mathbf{i = 0}$

As already mentioned above, by $v = \mathit{Verify}(X, K, q) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$, we know that there exists at least one successor $\mathit{Check}(\gamma, K, q = q_1) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ by Lemma 18 (1.). Let u be such a successor. If there is more than one, we can pick an arbitrary successor. Then, we define $L_0 = \{u\}$ and we get $\mathit{state}(L_0) = \{q\} = I_0$. Clearly, the first invariant is satisfied. The second invariant holds as well because I_0 is fixed to only contain q.

$\mathbf{i \to i + 1}$

We need to define $z_{i+1}(q_i)$ for each $q_i \in I_i$. From invariant (a), we get that $I_i = \mathit{state}(L_i)$ and thus there exists $u = \mathit{Check}(\gamma_{[i+1,n]}K, q_i) \in L_i$. We need to distinguish between the cases $\gamma_{i+1} \in T$ and $\gamma_{i+1} \in N$.

a) $\gamma_{i+1} \in T$

We depict u and its unique successor u' in Figure 43. As $\gamma_{i+1} \in T$, u' is of the form $\mathit{Check}(\gamma_{[i+2,n]}, K, q_{i+1})$ s.t. $q_i \xrightarrow{\gamma_{i+1}} q_{i+1}$. We define

$$z_{i+1}(q_i) = \{q_{i+1}\}.$$

$$\begin{tikzcd} u = \mathit{Check}(\gamma_{[i+1,n]}, K, q_i) \arrow[d] \\ u' = \mathit{Check}(\gamma_{[i+2,n]}, K, q_{i+1}) \end{tikzcd}$$

Figure 43.: Construction of L_{i+1} .

Thus, we map q_i to the clause containing only q_{i+1}. This map matches the desired definition in Equation 0.3 as $\sigma^{(k)}_{q_i,\gamma_{i+1}} = \{\{q_{i+1}\}\}$ (recall that $\sigma^{(k)}_{p,a}$ is defined to only contain a single clause $\{p'\}$ s.t. $p \xrightarrow{a} p'$).

It remains to define L_{i+1} and show that the invariants hold. As $u \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$, the same must hold for the unique successor: $u' \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ (Lemma 18 (1.)). We define

$$L_{i+1} = \{u' \mid u' \text{ successor of } u \in L_i\}$$

We argued above that the successors u' are contained in the attractor, therefore the second invariant is satisfied. For the first invariant, note that

$$\begin{aligned}
I_{i+1} &= \bigcup_{p \in I_i} z_{i+1}(p) \\
&= \bigcup_{p \in \mathit{state}(I_i)} z_{i+1}(p) \\
&= \bigcup_{p \in \mathit{state}(I_j)} p' \text{ where } p \xrightarrow{\gamma_{i+1}} p' \\
&= \mathit{state}(L_{i+1}),
\end{aligned}$$

which proves that the first invariant holds.

b) $\gamma_{i+1} \in N$

$$\begin{tikzcd}[column sep=small]
 & & u = \mathit{Check}(\gamma_{[i+1,n]}, K, q_i) \arrow[dl] \arrow[d, "\cdots"'] & \\
 & \mathit{Claim}(\dots) & cl = \mathit{Claim}(\gamma_{[i+1,n]}, P_{q_i}, K, \rho_i) \arrow[dll] \arrow[d] \arrow[dr, "\cdots"'] & \\
 v' = \mathit{Verify}(\gamma_{i+1}, P_{q_i}, q_i) & & \underbrace{\mathit{Check}(\dots) \quad u' = \mathit{Check}(\gamma_{[i+2,n]}, K, q_{i+1})}_{\text{one check node for each } q_{i+1} \in P_{q_i}} &
\end{tikzcd}$$

Figure 44.: Construction of L_{i+1}.

In this case, the successors of u are claim-nodes of the form $\mathit{Claim}(\gamma_{[i+1,n]}, P, K, q_i)$. By invariant (2) $u \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$, therefore at least one succeeding claim-node is also contained in the attractor $\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ (Lemma 18 (1.)). Let $cl = \mathit{Claim}(\gamma_{[i+1,n]}, P_{q_i}, K, q_i)$ be an arbitrary node among these claim-nodes. Figure 44 depicts the situation.

Let $v' = \mathit{Verify}(\gamma_{i+1}, P_{q_i}, q_i)$ be the verify-node that succeeds cl. From $cl \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$, it follows by Lemma 18(3.) that $v' \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$. Using the induction hypothesis of Lemma 19, we get that $P_{q_i} \in \sigma^{(k)}_{q_i,\gamma_{i+1}}$. Thus, the following definition is as desired in Equation 0.3.

$$z_{i+1}(q_i) = P_{q_i}$$

It remains to define L_{i+1} and show that the invariants hold. From $cl \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ and Lemma 18(3.), we get that all succeeding check-nodes $u' = \mathit{Check}(\gamma_{[i+2,n]}, K, q_{i+1})$ $(q_{i+1} \in P)$ are contained in the attractor $\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. We define

$$\begin{aligned}
L_{i+1} = \bigcup_{u \in L_i} \{u' = \mathit{Check}(\gamma_{[i+2,n]}, K, q_{i+1}) \mid \\
& u = \mathit{Check}(\gamma_{[i+1,n]}, K, q_i), \\
& cl = \mathit{Claim}(\gamma_{[i+1,n]}, P_{q_i}, K, q_i) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc} \text{ successor of } u \\
& \text{and } u' \text{ successor of } cl\}
\end{aligned}$$

which by the discussion above satisfies the second invariant.

For the first invariant consider

$$\begin{aligned} I_{i+1} &= \bigcup_{q_i \in I_i} z_{i+1}(q_i) \\ &= \bigcup_{q_i \in state(L_i)} z_{i+1}(q_i) \\ &= \bigcup_{q_i \in state(L_i)} P_{q_i} \\ &= \bigcup_{q_i \in state(L_i)} \bigcup_{p \in P_{q_i}} p = state(L_{i+1}) \end{aligned}$$

where P_{q_i} is the prediction of a claim-node $cl = Claim(\gamma_{[i+1,n]}, P_{q_i}, K, q_i) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ succeeding $u = Check(\gamma_{[i+1,n]}, K, q_i) \in L_i$. The invariant follows.

The resulting functions $z_1, \ldots, z_n$ are defined as desired in Equation 0.3. Thus, the functions construct a clause

$$K' = I_n = \{ \bigcup_{q_1 \in z_1(q_0)} \bigcup_{q_2 \in z_2(q_1)} \cdots \bigcup_{q_{n-1} \in z_{n-1}(q_{n-2})} z_n(q_{n-1})\} \in \sigma^{(k+1)}_{q,A}.$$

Correctness: Now it remains to show that indeed $K' = I_n \overset{!}{=} K$.

We prove this by mutual inclusion.

"$\subseteq$"

Let $p \in I_n$. Then, also $p \in state(L_n)$. Thus, there exists some $u = Check(\varepsilon, K, p)$ in L_n. From invariant (2), we get that $u \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. Therefore, the unique successor u' of u is a test-node $Test(p, K) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ (Lemma 18 (1.)) and thus $p \in K$ follows.

"$\supseteq$"

This direction is more involved. We have to show that for each $p \in K$, we have $p \in I_n = state(L_n)$. To show this, we use the inclusion minimality of K. If $p \notin state(L_n)$, then we can show that at least one successor of $Verify(A, K \setminus \{p\}, q)$ is contained in $\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. This contradicts the fact that $v = Verify(A, K, q)$ is contained in the antichain-attractor, because in this case v can not be $\preceq$-minimal (Lemma 18 (4.)). The reason why this holds is actually related to the reason why we can use an antichain-attractor in the first place. We make use of the fact that parts of the game graph reachable from $Verify(A, K, q)$ resp. $Verify(A, K \setminus \{p\}, q)$ are similar in structure and the predictions K resp. $K' = K \setminus \{p\}$ are only relevant in the test-nodes. If $p \notin state(L_n)$, then there will be no test-node of shape $Test(p, K)$ succeeding the nodes in L_n. Thus, it would be sufficient to have K' as prediction instead of K, which contradicts the fact that K is inclusion minimal.

Lemma 20
$\exists p \in K$ s.t. $p \notin state(L_n) \Rightarrow$ there exists a successor u of $Verify(A, K' = K \setminus \{p\}, q)$ contained in $\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$.

Proof. From $v \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$, it follows that there exists a successor $Check(\gamma, K, q) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ (Lemma 18 (1.)). We now make use of the similar structure of the parts of $\mathcal{G}_{\mathrm{GC}}$ that are reachable from v resp. v'. We can deduce that $Verify(A, K', q)$ has a successor of similar shape $\mathit{Check}(\gamma, K', q)$. By further use of the similar structures, we can prove the following statement.

$$\mathit{Check}(\gamma_{[i+1,n]}, K', q_i) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc} \text{ for } q_i \in \mathit{state}(L_i).$$

In particular, we get that also $\mathit{Check}(\gamma, K', q) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ as desired. We conduct a proof by reverse induction on i.

Base case $\mathbf{i = n}$:

$$\begin{tikzcd}
\mathit{Check}(\varepsilon, K, q_n) \arrow[d] & \rightsquigarrow & u = \mathit{Check}(\varepsilon, K', q_n) \arrow[d] \\
\mathit{Test}(q_n, K) & & t = \mathit{Test}(q_n, K')
\end{tikzcd}$$

Figure 45.

Let $q_n \in \mathit{state}(L_n)$. Then, there exists $\mathit{Check}(\varepsilon, K, q_n) \in L_n$. From the second invariant of the layer construction, we get that $\mathit{Check}(\varepsilon, K, q_n) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$, meaning that $q_n \in K$. Consider now $u = \mathit{Check}(\varepsilon, K', q_n)$. Its unique successor is test-node $t = \mathit{Test}(q_n, K')$ (Figure 45). As $q_n \neq p$ ($p \notin \mathit{state}(L_n)$), we know that $q_n \in K' = K \setminus \{p\}$ and thus both t and u are contained in the attractor (Lemma 18 (2.)).

Induction step $\mathbf{i \to i-1}$:

Let $q_{i-1} \in \mathit{state}(L_{i-1})$. We want to show that $u = \mathit{Check}(\gamma_{[i,n]}, K', q_{i-1}) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. We need to distinguish two cases.

- $\gamma_i \in T$

$$\begin{tikzcd}
\mathit{Check}(\gamma_{[i,n]}, K, q_{i-1}) \arrow[d] & \rightsquigarrow & u = \mathit{Check}(\gamma_{[i,n]}, K', q_{i-1}) \arrow[d] \\
\mathit{Check}(\gamma_{[i+1,n]}, K, q_i) & & u' = \mathit{Check}(\gamma_{[i+1,n]}, K', q_i)
\end{tikzcd}$$

Figure 46.

  By definition of $\mathit{state}(L_{i-1})$, there exists $\mathit{Check}(\gamma_{[i,n]}, K, q_{i-1}) \in L_{i-1}$ and by definition of L_i, its unique successor $\mathit{Check}(\gamma_{[i+1,n]}, K, q_i) \in L_i$. Thus, $q_i \in \mathit{state}(L_i)$. Then, we get from the induction hypothesis that $u' = \mathit{Check}(\gamma_{[i+1,n]}, K', q_i) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ and then, by Lemma 18 (2.), also $u \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$.

- $\gamma_i \in N$

  Again by definition of $\mathit{state}(L_{i-1})$, there exists $\mathit{Check}(\gamma_{[i,n]}, K, q_{i-1}) \in L_{i-1} \subseteq \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. For the construction of L_i, we picked one of the claim-nodes $\mathit{Claim}(\gamma_{[i,n]}, P, K, q_{i-1})$ succeeding the check-node in the attractor $\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. Then, all the check-nodes that succeeded this claim-node were incorporated into L_i. Refer to Figure 47 for a visual representation of the situation.

  We need to show that $u = \mathit{Check}(\gamma_{[i,n]}, K', q_{i-1}) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. From the structure that we described above, we get that u has a successor $\mathit{cl} = \mathit{Claim}(\gamma_{[i,n]}, P, K', q_{i-1})$ We prove that $\mathit{cl} \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$, then $u \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ follows by Lemma 18 (2.). By Lemma 18(3.) we need to show that

  a) the verify-node $u' = \mathit{Verify}(\gamma_i, P, q_{i-1})$ succeeding cl is contained in $\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$

$$\begin{tikzcd}[column sep=small]
 & \mathit{Check}(\gamma_{[i,n]}, K, q_{i-1}) \ \in L_{i-1} \arrow[dl] \arrow[d, "\dots"'] & \\
\mathit{Claim}(\dots) & \mathit{Claim}(\gamma_{[i,n]}, P, K, q_{i-1}) \ \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc} \arrow[dl] \arrow[d, "\dots"] \arrow[dr] & \\
u' = \mathit{Verify}(\gamma_i, P, q_{i-1}) \ \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc} & \mathit{Check}(\dots) & \mathit{Check}(\gamma_{[i+1,n]}, K, q_i) \ \in L_i \\
 & \underbrace{\qquad\qquad\qquad\qquad\qquad}_{\text{one check node for each } q_i \in P} &
\end{tikzcd}$$

$$\begin{tikzcd} {} \arrow[d, squiggly] \\ {} \end{tikzcd}$$

$$\begin{tikzcd}[column sep=small]
 & u = \mathit{Check}(\gamma_{[i,n]}, K', q_{i-1}) \arrow[dl] \arrow[d, "\dots"'] & \\
\mathit{Claim}(\dots) & \mathit{cl} = \mathit{Claim}(\gamma_{[i,n]}, P, K', q_{i-1}) \arrow[dl] \arrow[d, "\dots"] \arrow[dr] & \\
u' = \mathit{Verify}(\gamma_j, P, q_{i-1}) & \mathit{Check}(\dots) & u_{q_i} = \mathit{Check}(\gamma_{[i+1,n]}, K', q_i) \\
 & \underbrace{\qquad\qquad\qquad\qquad\qquad}_{\text{one check node for each } q_i \in P} &
\end{tikzcd}$$

Figure 47.: Extract of $\mathcal{G}_{\mathrm{GC}}$.

b) all check-nodes $u_{q_i} = \mathit{Check}(\gamma_{[i+1,n]}, K', q_i)$ succeeding cl are contained in $\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$.

The verify-node u' is actually the same verify-node succeeding $\mathit{Claim}(\gamma_{[i,n]}, P, K, q_{i-1})$ from above. As $\mathit{Claim}(\gamma_{[i,n]}, P, K, q_{i-1}) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$, $u' \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ by Lemma 18 (3.).

By construction of L_i, all check-nodes $\mathit{Check}(\gamma_{[i+1,n]}, K, q_i)$ are contained in L_i. This means that $q_i \in \mathit{state}(L_i)$. Therefore, we can apply the induction hypothesis to the check nodes $u_{q_i} = \mathit{Check}(\gamma_{[i+1,n]}, K', q_i)$ and get that $u_{q_i} \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$.

□

We finally get $K' = I_n = K$, which proves the correctness of the layer construction.

This closes the case of $X \in \bigcirc$ and we can tackle the case where X belongs to prover.

2. $\boldsymbol{X \in \Box}$

In this case, all the successors $\mathit{Check}(\gamma, K, q)$ of v have to be contained in the attractor $\overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. For each $\mathit{Check}(\gamma, K, q)$, we define functions $z_1^{(\gamma)}, \dots, z_n^{(\gamma)}$ ($n = |\gamma|$) with

$$\begin{aligned} z_i^{(\gamma)} :& I_{i-1}^{(\gamma)} \to \bigcup_{p \in Q} \sigma^{(k)}_{p,\gamma_i} \\ & p \mapsto C_p^{(\gamma_i)} \in \sigma^{(k)}_{p,\gamma_i}. \end{aligned} \tag{0.4}$$

These functions construct a clause

$$K' = \left\{ \bigcup_{X \to \gamma} K^{(\gamma)} \mid K^{(\gamma)} = \left\{ \bigcup_{p_1 \in z_1^{(\gamma)}(q)} \ \bigcup_{q_2 \in z_2^{(\gamma)}(q_1)} \ \dots \bigcup_{q_{n-1} \in z_{n-1}^{(\gamma)}(q_{n-2})} z_n^{(\gamma)}(q_{n-1}) \right\} \right\} \in \sigma^{(k+1)}_{q,X}$$

Finally, we will show that K and K' coincide.

We can make the same layer construction as for the case $X \in \bigcirc$ for each $\mathit{Check}(\gamma, K, q)$. The process defines $L_1^\gamma, \dots, L_n^\gamma$ resp. $z_1^{(\gamma)}, \dots, z_n^{(\gamma)}$ for $n = |\gamma|$ as required in Equation 0.4 . As in the case for $X \in \bigcirc$, the invariants hold:

a) $\mathit{state}(L_i^{(\gamma)}) = I_i^{(\gamma)} = \mathit{img}(z_i^{(\gamma)})$

b) $L_i^{(\gamma)} \subseteq \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$ for $i = 0, \dots, n$.

Let now

$$K'^{(\gamma)} = I_n^{(\gamma)} = \Big\{ \bigcup_{p_1 \in z_1^{(\gamma)}(q)} \quad \bigcup_{p_2 \in z_2^{(\gamma)}(p_1)} \quad \cdots \quad \bigcup_{p_{n-1} \in z_{n-1}^{(\gamma)}(p_{n-2})} \quad z_n^{(\gamma)}(p_{n-1}) \Big\}$$

and we show that $K \overset{!}{=} K' = \bigcup\limits_{A \to \gamma} K'^{(\gamma)} \in \sigma_{q,X}^{(k+1)}$.

We prove the claim by mutual inclusion.

"$\supseteq$"

We show that $K'^{(\gamma)} \subseteq K$ for all rules $X \to \gamma$. Let $p \in K'^{(\gamma)}$. Then, also $p \in I_n^{(\gamma)} = \mathit{state}(L_n^{(\gamma)})$. Thus, there exists some $u = \mathit{Check}(\varepsilon, K, p)$ in L_n^γ. From invariant b), we get that $u \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$. Therefore, the successor $\mathit{Test}(p, K) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$ and $p \in K$ follows.

"$\subseteq$"

Let $p \in K$. We need to show that $p \in \bigcup\limits_{X \to \gamma} K'^{(\gamma)} = \bigcup\limits_{X \to \gamma} I_n^{(\gamma)}$ for $n = |\gamma|$. We use the inclusion minimality of K again. Suppose therefore that $p \notin \bigcup\limits_{X \to \gamma} I_n^{(\gamma)} = \bigcup\limits_{X \to \gamma} \mathit{state}(L_n^{(\gamma)})$. This means that $p \notin I_n^{(\gamma)}$ for any γ. Like in the case of $X \in \bigcirc$, we prove that then a successor of $\mathit{Verify}(X, K' = K \setminus \{p\}, q)$ will be contained in $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$. Then, v can not be $\preceq$-minimal and thereby not contained in the antichain-attractor (Lemma 18 (4.)).

We proceed as in Lemma 20 and prove that

$$\mathit{Check}(\gamma_{[i+1,n]}, K', q_i) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)} \text{ for } q_i \in \mathit{state}(L_i^{(\gamma)})$$

by reverse induction for all γ.

We can reuse the proof from Lemma 20, but need to adapt the base case. Let therefore $q_n \in \mathit{state}(L_n^{(\gamma)} a)$. Thus, there exists a node $\mathit{Check}(\varepsilon, K, q_n) \in L_n^{(\gamma)}$ which is also contained in the attractor (invariant b)). Thus, the succeeding test-node $\mathit{Test}(q_n, K)$ is also contained in $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$ (Lemma 18 (1.)), meaning that $q_n \in K$. As $q_n \neq p$ ($q_n \in \mathit{state}(L_n) = I_n^{(\gamma)}, p \notin I_n^{(\gamma)}$), we know that $q_n \in K' = K \setminus \{p\}$. Therefore, node $\mathit{Test}(q_n, K')$ belongs to refuter and both the test-node and its predecessor $\mathit{Check}(\varepsilon, K', q_n)$ are contained in $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$ (Lemma 18 (2.)).

The rest of the proof is identical to the proof in Lemma 20 for each γ (using $z_i^{(\gamma)}$ instead of z_i).

It follows that $\mathit{Check}(\gamma, K', q) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$ for all $X \to \gamma$. Thus, $v = \mathit{Verify}(A, K, q)$ can not be $\preceq$-minimal and $v \notin \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$.

This proves the inclusion.

Having considered both cases $X \in \bigcirc$ and $X \in \square$, this concludes the proof of Lemma 19. $\square$

Lemma 21
Let $X \in N$.
$K \in \sigma^{(k)}_{q,X}$ *(reduced)* $\Rightarrow$ *Verify*$(X, K, q) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ *given that Verify*$(X, K, q) \in \mathcal{V}$.

Proof. We show the claim by induction over k.

Base case: $(k = 0)$
As for Lemma 19.
Induction step: $(k \to k + 1)$
Let therefore be $K \in \sigma^{(k+1)}_{q,X}$. We need to show that $v =$ *Verify*$(X, K, q) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. As K is of different shape depending on the owner of X, we distinguish two cases.

1. $\boldsymbol{X \in \bigcirc}$

   From the set representation of $\sigma^{(k+1)}_{q,X}$, we get that there exists a grammar rule $X \to_G \gamma$ s.t. K is of the following form.

$$K = I_n = \{ \bigcup_{p_1 \in z_1(p_0)} \quad \bigcup_{p_2 \in z_2(p_1)} \quad \cdots \quad \bigcup_{p_{n-1} \in z_{n-1}(p_{n-2})} z_n(p_{n-1})\}$$

   for some $z_1, \ldots, z_n$ with

$$z_i : I_{i-1} \to \bigcup_{p \in Q} \sigma^{(k)}_{p,\gamma_i}$$
$$p \mapsto C^{(\gamma_i)}_p \in \sigma^{(k)}_{p,\gamma_i}$$

   Note that we assume that $\sigma^{(k)}_{p,\gamma_i}$ are already reduced.

   The idea is to use the images $I_1, \ldots, I_n$ of the functions z_i to prove Lemma 22 below. From this lemma follows in particular that *Check*$(\gamma, K, q) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. Then, by Lemma 18 (4.), it follows that $v \in$ *IterateAttr*$(\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}, 1)$. We can prove that than also $v \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. Towards a contradiction, assume there exists node $v' =$ *Verify*$(X, K', q) \in$ *IterateAttr*$(\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}, 1)$ s.t. $v' \preceq v$. If there are several such nodes, we take the $\preceq$-minimal one. Then, $v' \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ and we can use Lemma 19 to get that $K' \subseteq K \in \sigma^{(k+1)}_{q,X}$. But then K could not have been contained in the reduced formula $\sigma^{(k+1)}_{q,X}$.

   Thus, the claim that $v \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ follows.

   Lemma 22
   For $i = 1, \ldots, n + 1$, it holds that

$$\textit{Check}(\gamma_{[i,n]}, K, q_{i-1}) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc} \textit{ for } q_{i-1} \in I_{i-1}.$$

   Proof. We prove this by reverse induction over i. Let $u =$ *Check*$(\gamma_{[i,n]}, K, q_{i-1})$ be the check-node of interest.

   Base case: $i = n + 1$

   In this case, u is of form *Check*(ε, K, q_n). Its unique successor in $\mathcal{G}_{\mathrm{GC}}$ is the test-node $t =$ *Test*(q_n, K)(see Figure 22). As we assumed that $q_n \in I_n$, we get that $q_n \in K$. Therefore, $t \in \overline{\mathsf{Attr}}^{(0)}_{\mathbb{A},\bigcirc} \subseteq \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. As t is no verify-node, it follows by Lemma 18(2.) that $u \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$.

$$\begin{tikzcd} u = \textit{Check}(\varepsilon, K, q_n) \arrow[d] \\ t = \textit{Test}(q_n, K) \end{tikzcd}$$

Figure 48.

Induction step: $(i \to i-1)$

We have to consider two cases, one for $\gamma_i \in T$ and one for $\gamma_i \in N$.

- $\gamma_i \in T$.

  In this case, u is of the form $\mathit{Check}(a\gamma_{[i+1,l]}, K, q_{i-1})$ with $q_{i-1} \in I_{i-1}$. As $\gamma_i \in T$, the (unique) successor of u is $u' = \mathit{Check}(\gamma_{[i+1,l]}, K, q_i)$ with $q_{i-1} \xrightarrow{a} q_i$ (Figure 22). From $\gamma_i \in T$ it also follows that $\sigma^{(k)}_{q_{i-1},\gamma_i}$ contains exactly one clause, namely $\{q \mid q_{i-1} \xrightarrow{a} q\}$. Making use of the fact that the automaton A is deterministic, we get that $q = q_i$ and thus $q_i \in I_i$. It follows by the induction hypothesis that $u' \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ and thus by Lemma 18(2.) also $u \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$.

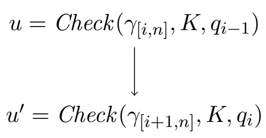
  
  $$\begin{tikzcd} u = \mathit{Check}(\gamma_{[i,n]}, K, q_{i-1}) \arrow[d] \\ u' = \mathit{Check}(\gamma_{[i+1,n]}, K, q_i) \end{tikzcd}$$

  Figure 49.

- $\gamma_i \in N$.

  In this case, u has multiple successor nodes, more precisely one claim-node per possible prediction $P \subseteq 2^Q$ (see Figure 50).

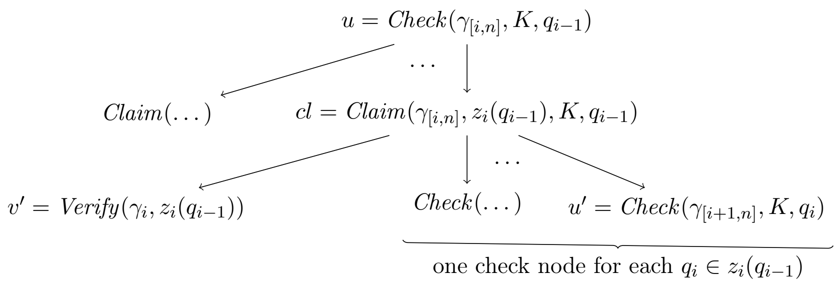
  
  $$\begin{tikzcd} & u = \mathit{Check}(\gamma_{[i,n]}, K, q_{i-1}) \arrow[dl] \arrow[d, "\cdots"'] & \\ \mathit{Claim}(\dots) & cl = \mathit{Claim}(\gamma_{[i,n]}, z_i(q_{i-1}), K, q_{i-1}) \arrow[dl] \arrow[d] \arrow[dr, "\cdots"'] & \\ v' = \mathit{Verify}(\gamma_i, z_i(q_{i-1})) & \mathit{Check}(\dots) & u' = \mathit{Check}(\gamma_{[i+1,n]}, K, q_i) \end{tikzcd}$$

  Figure 50.

  Thus, there will also be a succeeding claim-node with $P = z_i(q_{i-1}) \subseteq 2^Q$ namely $cl = \mathit{Claim}(\gamma_{[i,n]}, z_i(q_{i-1}), K, q_{i-1})$. We consider this claim-node and show that $cl \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. Then also $u \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ by Lemma 18(2.).

  Let us first investigate the verify-node $v' = \mathit{Verify}(\gamma_i, z_i(q_{i-1}), q_{i-1})$. By definition of the functions z_i, $z_i(q_{i-1}) \in \sigma^{(k)}_{q_{i-1},\gamma_i}$ (reduced). By applying the induction hypothesis of the theorem ($\gamma_i \in N$), we get that $v' = \mathit{Verify}(\gamma_i, z_i(\rho_{i-1})) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$.

  Now we consider the succeeding check-nodes. There is one check-node $u' = \mathit{Check}(\gamma_{[i+1,n]}, K, q_i)$ for $q_i \in z_i(q_{i-1})$. Then, it follows by the induction hypothesis that $u' \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$.

  As $v \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ and all check-nodes $u' \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$, it follows by Lemma 18(3.) that also $cl \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$ and thereby $u \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$.

  □

This concludes the case $X \in \bigcirc$.

2. $\boldsymbol{X \in \Box}$

   Let $K \in \sigma_{k+1}A$. Because $X \in \Box$, we can write K as

$$K = \left\{ \bigcup_{X \to \gamma} K^{(\gamma)} \mid K^{(\gamma)} = \left\{ \bigcup_{p_1 \in z_1^{(\gamma)}(q)} \quad \bigcup_{q_2 \in z_2^{(\gamma)}(q_1)} \quad \cdots \quad \bigcup_{q_{n-1} \in z_{n-1}^{(\gamma)}(q_{n-2})} z_n^{(\gamma)}(q_{n-1}) \right\} \right\}$$

for some $z_1^{(\gamma)}, \ldots, z_n^{(\gamma)}$ with

$$\begin{aligned} z_i^{(\gamma)} : & I_{i-1}^{(\gamma)} \to \bigcup_{p \in Q} \sigma_{p,\gamma_i}^{(k)} \\ & p \mapsto C_p^{(\gamma_i)} \in \sigma_{p,\gamma_i}^{(k)} \end{aligned}$$

where $n = |\gamma|$ for each γ s.t. $X \to \gamma$.

To prove that $v = \mathit{Verify}(X, K, q) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$, we first show that all its successor nodes are included in the attractor $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$. Then, by Lemma 18(4.), $v \in \mathit{IterateAttr}(\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}, 1)$. But we argued in the case for $X \in \bigcirc$ that in this case also $v \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$ follows.

All succeeding nodes are check-nodes of shape $\mathit{Check}(\gamma, K, q)$, one for each rule $X \to \gamma$.

In order to show that the check-nodes are included in $\overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$, we use the following lemma which is similar to Lemma 22.

Lemma 23
Let γ be s.t. $X \to \gamma$. For $i = 1, \ldots, n$, it holds that

$$\mathit{Check}(\gamma_{[i,n]}, K, q_{i-1}) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)} \text{ for } q_{i-1} \in I_{i-1}^{(\gamma)}.$$

Proof. The proof is conducted similarly to the proof of Lemma 22 by reverse induction. Only the base case needs a slight adaptation, the induction step is the same (use $z_i^{(\gamma)}$ instead of z_i).

Let thus be $i = n + 1$ and $u = \mathit{Check}(\varepsilon, K, q_n)$ with $q_n \in I_n^{(\gamma)}$. Node u has an edge to test-node $t = \mathit{Test}(q_n, K)$. From the assumption, we get that $q_n \in I_n^{(\gamma)} = K^{(\gamma)} \subseteq K$. Thus $t \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$ and therefore also $u \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$ by Lemma 18 (2.). □

This concludes the proof of Lemma 21. □

With the two Lemmata at hand, we can finally reestablish the condition that the formulas are reduced and thus show Theorem 9. Recall the claim:
Let $X \in N$.

1. If $\mathit{Verify}(X, K, q) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$ then $K \in \sigma_{q,X}^{(k)}$ (reduced).

2. If $K \in \sigma_{q,X}^{(k)}$ (reduced) then $\mathit{Verify}(X, K, q) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$ given that $\mathit{Verify}(X, K, q) \in \mathcal{V}$.

Proof.

1. Let K be s.t. $\mathit{Verify}(X, K, q) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$. By Lemma 19, we get that $K \in \sigma_{q,X}^{(k)}$. Assume now that there exists $K' \subset K$ in $\sigma_{q,X}^{(k)}$. Then, by Lemma 21, $\mathit{Verify}(X, K', q) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k+1)}$ contradicting the fact that the attractor contains only antichains.

2. Directly be Lemma 21.

□

Despite the non-symmetric directions of the statement of Theorem 9, it suggests a strong relation between the formula $\sigma_{q,X}$ and the verify-nodes $\mathit{Verify}(X, K, q)$ in the antichain attractor. We already hinted the intuition behind the result. If formula $\sigma_{q,X}$ contains clause K, refuter can enforce the derivation of a terminal word w from X s.t. $q \xrightarrow{w} p \in K$. On the other hand, refuter wins from node $v = \mathit{Verify}(X, K, q)$ if the prediction K is correct for X and q, i.e. refuter can enforce the derivation of a terminal word w with $q \xrightarrow{w} p \in K$. As refuter wins from v, the node is contained in the attractor. The $\preceq$-minimality of v is guaranteed by the fact that we only consider clauses from the reduced formula $\sigma_{q,X}$ and vice versa.

However, this does not yet explain why the intermediary formulas $\sigma^{(k)}_{q,X}$ correspond to the verify-nodes $\mathit{Verify}(X, K, q)$ in the k-th attractor set $\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$. Recall that, if $K \in \sigma^{(k)}_{q,X}$, refuter can enforce a play s.t. it ends in a terminal word w with $q \xrightarrow{w} p \in K$ and that has at most k applications of rules in each branch of the parse tree. We will argue that the same holds if $\mathit{Verify}(X, K, q) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$. But before we explain why this is the case, we describe how the plays are composed in the game graph $\mathcal{G}_{\mathrm{GC}}$.

In the proof of Theorem 9, we used the functions z_i of the set representation of $K \in \sigma^{(k)}_{q,X}$ to define a winning strategy for refuter from the verify-node $v = \mathit{Verify}(X, K, q) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$. More precisely, the z_i defined predictions for the claim-nodes that are encountered on the paths conform to the winning strategy. The predictions are s.t. refuter would win the verify-branch following each claim-node. Thus, we only had to consider the skip-branches. But for them, we had to take every possible choice of prover into account. Therefore, the z_i define a tree rooted at v consisting of only check- and claim-nodes. All the nodes in the tree are contained in the attractor $\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$. We call the tree the **composition tree** of verify-node v. The successor(s) of the root are check-node(s) of shape $\mathit{Check}(\gamma, K, q)$ if there is a grammar rule $X \to_G \gamma$, depending on whether $X \in \bigcirc$ or $X \in \square$.

In general, the check-nodes in the tree are of shape $c = \mathit{Check}(\gamma_{[i,n]}, K, q')$ where $\gamma_{[i,n]}$ is a suffix of γ. The check-nodes c represent the composition of the plays starting from $\gamma_1, \dots, \gamma_{i-1}$ and q. The states q' in the check-nodes track the target states of the terminal words w that are derived by the composed plays from q on, i.e. $q \xrightarrow{w} q'$. Thus, the composition of the plays is executed by the paths in the composition tree.

However, we do not compose arbitrary plays starting from γ_i with each other. We only compose plays of γ_i s.t. the resulting (composed) plays end in a terminal word w with $q \xrightarrow{w} K$ and the parse trees of the plays have at most k applications of rules in each branch. This is possible because $\mathit{Verify}(X, K, q) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ (we comment on this below). Refer to Figure 51 for an example of such a composition tree and the corresponding composition of plays.

Although this way of composing plays by a tree seems very different to the composition by paths in the saturation method, they follow the same idea. The main difference is that in the composition tree of $\mathit{Verify}(X, K, q)$, we only consider the path $q \xrightarrow{\gamma}{}^{*}_{\mathscr{A}} K$ leading up to clause $K \in \sigma^{(k)}_{q,X}$. We actually get exactly this path if we merge all states of check-nodes in the same layer of the tree. This, the inner check-nodes of the tree coincide with the intermediary states of the path $q \xrightarrow{\gamma}{}^{*}_{\mathscr{A}} K$.

The formula $\sigma^{(k)}_{q,X}$ captures maximal plays from X that have at most k applications of grammar rules in each branch of their parse tree. By the following inductive reasoning, one can see that the same holds true for the verify-nodes $\mathit{Verify}(X, K, q) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$. In the attractor set $\overline{\mathsf{Attr}}^{(0)}_{\mathbb{A},\bigcirc}$, no verify-nodes are contained at all. Indeed, no maximal plays from any non-terminal X exist s.t. their parse tree has at most 0 application of rules in each branch. Suppose the claim holds true for the verify-nodes in $\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$. Let us consider a verify-node $v = \mathit{Verify}(X, K, q) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$. For the sake of simplicity, assume that $X \in \bigcirc$, the arguments for the case of $X \in \square$ are similar. Then, there exists a grammar rule $X \to_G \alpha$ s.t. $\mathit{Check}(\gamma, K, q)$ succeeds v in the composition tree. Consider now all $\gamma_i \in N$. For all

$$\begin{tikzcd}
 & & & & \mathit{Verify}(Z,K,q) \arrow[d] & & & & \\
 & & & & \mathit{Check}(XY,K,q) \arrow[d] & & & & \\
 & & \mathit{Verify}(X,G_1,q) & & \mathit{Claim}(XY,G_1,K,q) \arrow[ll] \arrow[dll] \arrow[drr] & & & & \\
 & & \mathit{Check}(Y,K,q_1) \arrow[d] & & & & \mathit{Check}(Y,K,q_2) \arrow[d] & & \\
\mathit{Verify}(Y,F_1,q_1) & & \mathit{Claim}(Y,F_1,K,q_1) \arrow[ll] \arrow[dl] \arrow[dr] & & & & \mathit{Claim}(Y,H_1,K,q_2) \arrow[rr] \arrow[dl] \arrow[dr] & & \mathit{Verify}(Y,H_1,q_2) \\
 & \mathit{Check}(\varepsilon,K,p_1) & & \mathit{Check}(\varepsilon,K,p_2) & & \mathit{Check}(\varepsilon,K,p_4) & & \mathit{Check}(\varepsilon,K,p_5) &
\end{tikzcd}$$

Figure 51.: Example of a composition tree and the corresponding composition of plays in the game graph $\mathcal{G}_{\mathrm{GC}}$. Suppose there exists a grammar rule $Z \rightarrow_G XY$ and that $Z \in \bigcirc$. Furthermore, let $v = \mathit{Verify}(Z, K, q) \in \overline{\mathsf{Attr}}^{(k+1)}_{\mathbb{A},\bigcirc}$. If we have the following formulas, the composition tree of v is of the shape depicted in the figure. (1) $K = \{p_1, p_2, p_4, p_5\} \in \sigma^{(k+1)}_{q,Z}$
(2) $\sigma^{(k)}_{q,X} = \{G_1, G_1\}$ where $G_1 = \{q_1, q_2\}, G_1 = \{q_3\}$
(3) $\sigma^{(k)}_{q_1,Y} = \{F_1, F_2\}$ where $F_1 = \{p_1, p_2\}, F_2 = \{p_3\}$
(4) $\sigma^{(k)}_{q_2,Y} = \{H_1, H_2\}$ where $H_1 = \{p_4, p_5\}, H_2 = \{p_6\}$

the claim-nodes $\mathit{Claim}(\gamma_i, K', K, q')$ it holds that they are contained in $\overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$. But then, by Lemma 18 (3.), the verify-nodes succeeding them are contained in $\overline{\mathsf{Attr}}^{(k-1)}_{\mathbb{A},\bigcirc}$. Thus, if we compose plays starting at $\gamma_1, \ldots, \gamma_n$ in the tree, we know that the composing plays have either at most $k-1$ applications in each branch of the parse tree (if $\gamma_i \in N$) or 0 (if $\gamma_i \in T$). Thus, the resulting play has a parse tree with at most k applications in each branch.

The result of Theorem 9 furthermore confirms the statement that we made in Subsection 9 about the system of equations of the alternative summary algorithm. We are only interested in the value of $\sigma_{q_0,S}$ for q_0 the starting state of the deterministic automaton A^{det} and S the initial symbol of S. All other formulas $\sigma^{(k)}_{q,X}$ for state q and non-terminal X are only used to compute $\sigma_{q_0,S}$. But as we compute the formulas in a bottom-up manner, we do not know which formulas are actually needed and which ones are superfluous. But as we discussed above, the game graph $\mathcal{G}_{\mathrm{GC}}$ is constructed top-down and excludes some verify-nodes $\mathit{Verify}(X, K, q)$ where no derivation process from S to $wX\alpha$ with $q_0 \xrightarrow{w} q$ exists. Unfortunately, not all superfluous verify-nodes are excluded and may still appear in the attractor.

9.2.2. Comparison: Box-based Summarization vs. Guess & Check

Suppose now that we used the box automaton $\mathcal{A}_M$ of A as base to construct the game graph $\mathcal{G}_{\mathrm{GC}}$. Then, we can apply Theorem 7, linking the formulas $\sigma^{(k)}_{\rho,X}$ to the formulas $\rho; \sigma^{(k)}_X$, to the statement of Theorem 9. Then, we get the following result.

Theorem 10
Let $X \in N$ be a non-terminal and $\rho \in B(A)$ a box.

1. *If $\mathit{Verify}(X, K, \rho) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ then $K \in \rho; \sigma^{(k)}_X$ (reduced).*

2. *If $K \in \rho; \sigma^{(k)}_X$ (reduced) then $\mathit{Verify}(X, K, \rho) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ given that $\mathit{Verify}(X, K, \rho) \in \mathcal{V}$.*

Although we again get a strong relation between the formulas and the verify-nodes in the attractor, the theorem also hints that the Guess & Check approach does not make use of the power of the boxes and computes the same information is several times.

The reason is the following. Suppose we have grammar rule $S \to_G XY$ and we want to compute the plays starting from S. For the sake of simplicity, assume that this is the only rule with left-hand side X. In the box-based summary approach, we compute the formula σ_S by composing the formulas σ_X and σ_Y with the composition operator. The formulas σ_X and σ_Y were computed independently of each other.

In the Guess & Check approach, we have to consider the composition trees of verify-nodes $\mathit{Verify}(S, K, \rho_\varepsilon)$ that are contained in the attractor. In each tree, we have one claim-node of shape $\mathit{Claim}(XY, K_1, K, \rho_\varepsilon)$ and one node $\mathit{Claim}(Y, K_2, K, \rho)$, $\rho \in K'$ for each branch created by the first claim-node. Each of the claim-nodes for Y has a succeeding verify-node $\mathit{Verify}(Y, K_2, \rho)$ for which we had to compute whether it is contained in the attractor in a previous step. From Theorem 9, we get that these K_2 are of shape $\rho; K'$ for a clause $K' \in \sigma_X$. In most cases, there will be some ρ, ρ' s.t. both $\mathit{Verify}(Y, \rho; K', \rho)$ and $\mathit{Verify}(Y, \rho'; K', \rho')$ for any $K' \in \sigma_X$ are contained in some (not necessarily the same) of the composition trees. In fact this always happens if σ_X either contains at least two clauses or one clause with at least two atomic propositions. Thus contrary to the summary approach, the Guess & Check method computes the information of the plays from Y dependent on the plays from X.

In the example of Figure 52, the verify-nodes $v_1 = \mathit{Verify}(Y, \{\rho_{ac}\} = \rho_a; \{\rho_c\}, \rho_a)$ and $v_2 = \mathit{Verify}(Y, \{\rho_{bc}\} = \rho_b; \{\rho_c\}, \rho_b)$ are even contained in the same composition tree. Both verify-branches are nearly identical. It would actually be possible to conclude from $v_1 \in \mathsf{Attr}_{\mathbb{A},\bigcirc}$ that also $v_2 \in \mathsf{Attr}_{\mathbb{A},\bigcirc}$ or vice versa if we made use of the box composition. Intuitively, $v_1 \in \mathsf{Attr}_{\mathbb{A},\bigcirc}$ means that refuter can enforce the derivation of a word w s.t. $\rho_a \xrightarrow{w} \rho_{ac} = \rho_a; \{\rho_c\}$. But then for word w also holds that $\rho_b \xrightarrow{w} \rho_b; \{\rho_c\} = \rho_{bc}$. Therefore, we could conclude $v_2 \in \overline{\mathsf{Attr}_{\mathbb{A},\bigcirc}}$.

9.2.3. Alternative Guess & Check algorithm

Although the Guess & Check algorithm that we defined in Section 5 can not make use of the power of the boxes, it is possible to adapt the algorithm s.t. it can make proper use of the boxes. The reason why this is possible is that the predictions are held very general, they can be any set of boxes. In the regular Guess & Check algorithm, the predictions that are made at claim-nodes $c = \mathit{Claim}(X\alpha, P', P, \rho)$ should capture the target state τ in $\mathcal{A}_M$ of the words w derivable from X from ρ on. In this setting, we only use the automaton $\mathcal{A}_M$ as a deterministic automaton that captures state changes of derivable words. However, we do not make use of the meaning of the paths in $\mathcal{A}_M$, namely that $\rho \xrightarrow{w} \tau$ if $\tau = \rho_w; \rho$.

However, with this property in mind, we can change the meaning of the prediction P' at claim-node c. Prediction P' should capture the target state ρ_w of the words w derivable from X from state ρ_ε on. In other words, it should capture the boxes of the derivable words. This means that the succeeding verify-node is of shape $\mathit{Verify}(X, P', \rho_\varepsilon)$ and does not depend on ρ anymore. As we are still interested in the state changes that the derivable words w induce from ρ on in A^{det}, we compose the boxes $\rho_w \in P'$ with ρ in the check-nodes $\mathit{Check}(\alpha, P, \rho; \rho_w)$ succeeding c. Actually, we get the same check-nodes if we used the prediction as in Section 5.

Formally, the game graph of the alternative Guess & Check algorithm is defined as follows.

Definition 29
Let $\mathcal{G}_{GC}$= ($\mathcal{V}$, $\mathcal{E}$, v_0) be the game graph constructed in the regular Guess & Check algorithm using the box automaton $\mathcal{A}_M$ as base. Then, the game graph $\mathcal{G}_{GC}$' = ($\mathcal{V}'$, $\mathcal{E}'$, v_0) of the alternative Guess & Check algorithm (based on $\mathcal{A}_M$ as well) is defined as follows.

- *The set of nodes $\mathcal{V}'$ contains only verify-nodes with state ρ_ε. Otherwise it is similar to $\mathcal{V}$, i.e. $\mathcal{V}' = \mathcal{V} \setminus \{\, \mathit{Verify}(X, P, \rho) \mid \rho \neq \rho_\varepsilon\}$*

$$\begin{tikzcd}[column sep=tiny]
 & & & \mathit{Verify}(S, K, \rho_\varepsilon) \arrow[d] & & & \\
 & & & \mathit{Check}(XY, K, \rho_\varepsilon) \arrow[d] & & & \\
 & & \mathit{Verify}(X, \{\rho_a, \rho_b\}, \rho_\varepsilon) & \mathit{Claim}(XY, \{\rho_a, \rho_b\}, K, \rho_\varepsilon) \arrow[l] \arrow[dll] \arrow[drr] & & & \\
 & \mathit{Check}(Y, K, \rho_a) \arrow[d] & & & & \mathit{Check}(Y, K, \rho_b) \arrow[d] & \\
 & \mathit{Claim}(Y, \{\rho_{ac}\}, K, \rho_a) \arrow[dl] \arrow[dr] & & & & \mathit{Claim}(Y, \{\rho_{bc}\}, K, \rho_b) \arrow[dl] \arrow[dr] & \\
v_1 = \mathit{Verify}(Y, \rho_a; \{\rho_c\}, \rho_a) \arrow[d] & & \mathit{Check}(\varepsilon, K, \rho_{ac}) \arrow[d] & & v_2 = \mathit{Verify}(Y, \rho_b; \{\rho_c\}, \rho_b) \arrow[d] & & \mathit{Check}(\varepsilon, K, \rho_{bc}) \arrow[d] \\
\mathit{Check}(c, \rho_a; \{\rho_c\}, \rho_a) \arrow[d] & & \mathit{Test}(\rho_{ac}, K) & & \mathit{Check}(c, \rho_b; \{\rho_c\}, \rho_b) \arrow[d] & & \mathit{Test}(\rho_{bc}, K) \\
\mathit{Check}(\varepsilon, \rho_a; \{\rho_c\}, \rho_a; \rho_c) \arrow[d] & & & & \mathit{Check}(\varepsilon, \rho_b; \{\rho_c\}, \rho_b; \rho_c) \arrow[d] & & \\
\mathit{Test}(\rho_a; \rho_c, \rho_a; \{\rho_c\}) & & & & \mathit{Test}(\rho_b; \rho_c, \rho_b; \{\rho_c\}) & &
\end{tikzcd}$$

Figure 52.: Example of the composition tree of $\mathit{Verify}(S, K, \rho_\varepsilon) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}$. In the example, we have grammar rules $S_\bigcirc \to_G XY$, $X_\Box \to_G a \mid b$ and $Y_\Box \to_G c$. For the sake of simplicity, we do not specify the automaton of the context-free game. We simply assume that $\rho_a \neq \rho_b$. Note that the verify-branches at v_1 resp. v_2 are nearly identical.

- *The set of transitions $\mathcal{E}'$ is similar to $\mathcal{E}$ except for the transitions with a claim-node as left-hand side. We redefine them by*

$$\mathit{Claim}(X\alpha, P', P, \rho) \xrightarrow{(\mathit{verify})} \mathit{Verify}(X, P', \rho_\varepsilon)$$
$$\mathit{Claim}(X\alpha, P', P, \rho) \xrightarrow{\mathit{skip}\ (\tau \in P')} \mathit{Check}(\alpha, P, \rho; \tau).$$

For the example in of Figure 52, the game graph $\mathcal{G}_{\mathrm{GC}}$' would have verify-nodes $v_1 = v_2 = \mathit{Verify}(Y, \rho_c, \rho_\varepsilon)$ succeeding the claim-nodes $\mathit{Claim}(Y, \{\rho_{ac}\}, K, \rho_a)$ resp. $\mathit{Claim}(Y, \{\rho_{bc}\}, K, \rho_b)$. Thus, the verify-branches following the two claim-nodes would coincide. The check-nodes succeeding the claim-nodes are analogue to the one in $\mathcal{G}_{\mathrm{GC}}$, i.e. $\mathit{Check}(\varepsilon, K, \rho_a; \rho_c) = \mathit{Check}(\varepsilon, K, \rho_{ac})$ resp. $\mathit{Check}(\varepsilon, K, \rho_{bc})$

The correctness proof of the alternative Guess & Check algorithm is analog to the correctness proof for the regular algorithm stated in Section 5.

For the comparison of the box-based summary algorithm with the alternative Guess & Check algorithm, we can make a similar statement as in Theorems 9 and 10. We leave the theorem without proof, but it is similar to the proof of Theorem 10.

Theorem 11
Let $X \in N$ be a non-terminal and $\rho \in B(A)$ a box.

1. *If $\mathit{Verify}(X, K, \rho_\varepsilon) \in \overline{\mathsf{Attr}}^{(k)}_{\mathbb{A},\bigcirc}$ then $K \in \sigma^{(k)}_X$ (reduced).*

2. *If* $K \in \sigma_X^{(k)}$ *(reduced) then* $\mathit{Verify}(X, K, \rho) \in \overline{\mathsf{Attr}}_{\mathbb{A},\bigcirc}^{(k)}$ *given that* $\mathit{Verify}(X, K, \rho) \in \mathcal{V}$.

Note that the restriction in the second direction is now less strong. If a non-terminal X is reachable from S i.e. there exists a derivation process from S to some $wX\alpha$, then the corresponding verify-node $\mathit{Verify}(X, K, \rho_\varepsilon)$ is contained in the game graph $\mathcal{G}_{\mathrm{GC}}$', independent of the terminal prefix w of $wX\alpha$. By a simple preprocessing on the grammar, we could eliminate the rules whose left-hand sides are unreachable from S. Then, we could drop the restriction in the theorem.

The statement of the theorem show that there is a one-to-one relation of the formulas $\sigma_X^{(k)}$ and the verify-nodes $\mathit{Verify}(X, K, \rho_\varepsilon)$. The predictions K in these nodes correspond to the clauses of the formula.

Although the alternative Guess & Check algorithm now makes use of the power of the boxes, it still can not avoid the upfront determinization of the automaton A into the box automaton $\mathcal{A}_M$. But this was the reason why we used the boxes in the summary algorithm. The alternative Guess & Check algorithm remains a top-down approach for which the whole state space of $\mathcal{A}_M$ needs to be known. As in most cases, the box automaton is much larger than the minimal determinized automaton recognizing $\mathcal{L}(A)$, the game graph $\mathcal{G}_{\mathrm{GC}}$' will be larger than a game graph $\mathcal{G}_{\mathrm{GC}}$ based on the minimal deterministic automaton, even though the game graph $\mathcal{G}_{\mathrm{GC}}$' may have less verify-nodes. Thus, it is preferable to use the minimal deterministic automaton in combination with the regular Guess & Check algorithm.

10. Conclusion

Recapitulation In the first part of the thesis, we introduced three approaches to solve context-free games: The summary, saturation and Guess & Check approach. For the latter two, we also proved the correctness of the algorithms as they were adapted from algorithms for pushdown game systems.

In the second part, we conducted a detailed comparison of the methods. We focused on three aspects.

First, we examined whether the approaches separate the task of computing the possible plays from the task of determining the winning ones (for refuter) among them. Both the summary and the saturation approach first compute the possible plays in form of formulas resp. a saturated automaton. Only after this step it is determined whether refuter can enforce a winning play, by evaluating the formula resp. deciding whether the saturated automaton accepts the starting symbol S of the grammar. In the Guess & Check approach however the game graph includes both information.

At the claim-nodes of shape $\mathit{Claim}(S, P, P_{\mathrm{rej}}, \rho_\varepsilon)$, refuter only wins if she can enforce a play ending in w with $\rho_w \in P$, which represent the possible plays, and if the derived words w are not accepted by the automaton. With the attractor, we check both conditions, the first in the verify-branch and the second in the skip-branches.

Second, we studied how the three approaches deal with a non-deterministic automaton on the right-hand side of the inclusion. Both the saturation and the Guess & Check approach do not get around an upfront determinization of the automaton as they need the whole state space of the automaton before at the beginning of their computations. The summary approach however determinizes the automaton on-the-fly along the terminal words that are derivable in the grammar. Key to this ability are the separate computation of the possible resp. winning plays, the bottom-up computation of the plays and the proper use of the power of the boxes.

Finally, we compared the intermediary information computed by the respective fixed-point iterations of the three approaches.

First, we related the formulas $\sigma^{(k)}_{q,X}$ of the alternative summary algorithm to the transitions $q \xrightarrow[k]{X}{}^{*}_{\mathscr{A}} K$ that were added to the alternating automaton during the saturation. The right-hand sides of the transitions correspond to the clauses K of the formulas $\sigma^{(k)}_{q,X}$. If the box automaton of A is used as base for the saturation algorithm, the right-hand sides of the transitions $\rho \xrightarrow[k]{X}{}^{*}_{\mathscr{A}} K$ correspond to the clauses of formula $\rho; \sigma^{(k)}_{X}$. This result furthermore showed that composition of plays by paths in the saturation algorithm does not allow to make use of the power of the boxes.

Second, we related the formulas $\sigma^{(k)}_{q,X}$ resp. $\sigma^{(k)}_{X}$ to the verify-nodes of shape $\mathit{Verify}(X, K, q)$ resp. $\mathit{Verify}(X, K, \rho)$ (if we use the box automaton as base for the construction of the game graph) that are contained in the k-th fixed-point approximants of the attractor. The predictions K correspond to the clauses of the formulas $\sigma^{(k)}_{q,X}$ resp. $\rho; \sigma^{(k)}_{X}$.

At the end, we presented an adaptation of the Guess & Check algorithm that makes use of the ;-operator to compose the plays. This allows us to only keep verify-nodes $\mathit{Verify}(X, K, \rho_\varepsilon)$ in the game graph.

Future Work For future work, we would first like to include the work of Ong and Kobayashi [6], and Vardi [7] into our comparison by adapting their algorithms to context-free games.

Second, we intend to extend the comparison to liveness synthesis, where the program template is given by a context-free game and the specification by a Büchi automaton. The liveness synthesis problem asks to solve whether prover can enforce the derivation of an ω-word that is accepted by the Büchi automaton. Actually, both the works of Cachat [1] and Walukiewicz [10] already consider pushdown game systems with a Büchi resp. parity automaton on the right-hand side of the inclusion. In [8], the summary algorithm has been lifted to liveness.

Finally, we would like to lift the left-hand side of the inclusion to higher-order recursion schemes and adapt the algorithms to solve the resulting synthesis problem.

Bibliography

[1] T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *ICALP*, volume 2, pages 704–715. Springer, 2002.

[2] T. Cachat. *Games on pushdown graphs and extensions.* PhD thesis, Bibliothek der RWTH Aachen, 2003.

[3] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* CUP, 1990.

[4] M. De Wulf, L. Doyen, T. A. Henzinger, and J. F. Raskin. Antichains: A new algorithm for checking universality of finite automata. In *Proceedings of the 18th International Conference on Computer Aided Verification*, CAV'06, pages 17–30, Berlin, Heidelberg, 2006. Springer-Verlag.

[5] L. Holík, R. Meyer, and S. Muskalla. Summaries for context-free games. In *FSTTCS*, volume 65 of *LIPIcs*, pages 41:1–41:16, 2016.

[6] N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. *2009 24th Annual IEEE Symposium on Logic In Computer Science*, 2009.

[7] O. Kupferman, N. Piterman, and M. Y. Vardi. An automata-theoretic approach to infinite-state systems. *Time for Verification Lecture Notes in Computer Science*, page 202–259, 2010.

[8] R. Meyer, S. Muskalla, and E. Neumann. Liveness verification and synthesis: New algorithms for recursive programs. *arXiv preprint arXiv:1701.02947*, 2017.

[9] A. Muscholl, T. Schwentick, and L. Segoufin. Active context-free games. *Lecture Notes in Computer Science STACS 2004*, page 452–464, 2004.

[10] I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and computation*, 164(2):234–263, 2001.

[11] M. Zimmermann, F. Klein, and A. Weinert. Infinite games lecture notes, 2016.