From Hyperedge Replacement Grammars to decidable Hyperedge Replacement Games

Christoph Peuser*

Carl von Ossietzky Universität Oldenburg, Germany
peuser@informatik.uni-oldenburg.de

Abstract. We consider correctness of hyperedge replacement grammars *under adverse conditions*. In contrast to existing approaches, the influence of an adverse environment is considered in addition to system behaviour. To this end, we construct a hyperedge replacement game where rules represent the moves available to players and a temporal condition specifies the desired properties of the system. In particular, the construction of parity pushdown games from hyperedge replacement grammars results in a decidable class of games.

Keywords: Context-Free Graph Grammars, Game Theory, Hyperedge Replacement Grammars, Pushdown Games, Parity Games

1 Introduction

Graph transformation systems and graph grammars [5] offer a graphical, yet precise formalism for modeling a system. The system state is a graph and state changes are modeled by graph transformations. There are a number of approaches to proving the correctness of a graph transformation system relative to conditions, e.g. for nested conditions [8, 13] or the μ-calculus [3].

We consider system correctness under *adverse conditions*, such as interference from the environment. Once we consider actions of the system and environment seperately, we require more control over the relative frequency with which system or environment may act. By modeling system and environment as players of a game, we limit the number of moves a player is allowed to make.

Games are a natural model for processes under the influence of an adverse environment and there are many results for solving these games, see e.g. [6]. In particular, solutions to parity games cover a variety of interesting properties and can be solved both in the case of finite state spaces and for some infinite state spaces, such as those that can be represented by a pushdown process [15], i.e. a pushdown automaton without acceptance features.

Graph grammars have an infinite state space in general and games defined over infinite state spaces are not generally decidable.

* This work is supported by the German Research Foundation through the Research Training Group (DFG GRK 1765) SCARE (www.scare.uni-oldenburg.de).

In this paper, we restrict ourselves to context-free hypergraph grammars, or more specifically hyperedge replacement (HR) grammars, and construct hyperedge replacement (HR) games to use the decidability result for parity pushdown games from Walukiewicz [15].

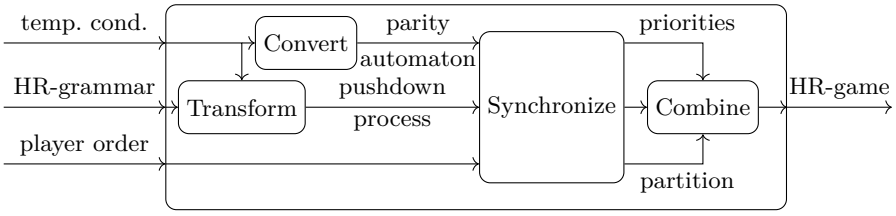


Fig. 1: Construction of a HR game

This process is illustrated in Fig. 1: Given an "ordered" HR-grammar, a temporal graph condition and a player order, we construct a HR-game as follows:

1. **Transform** the ordered HR-grammar and the temporal graph condition to a pushdown process, such that the states of the pushdown process are labelled with the atomic conditions of the temporal graph condition.
2. **Convert** the temporal graph condition into a parity automaton.
3. **Synchronize** the pushdown process with the player order and the parity automaton.
4. **Combine** the resulting pushdown process, partition function and priority function to form a HR-game.

The system is said to be correct, if there is a winning strategy for the system player. By the results of Walukiewicz [15], we can find such a strategy, if the HR-game is a parity pushdown game.

The remainder of the paper is structured as follows: Section 2 introduces ordered hyperedge replacement grammars and temporal graph conditions. Section 3 provides a definition for pushdown processes and parity pushdown games. Section 4 presents the transformation of an ordered hyperedge replacement grammar and a temporal graph condition to a pushdown process. In Section 5 a hyperedge replacement game is constructed from the pushdown process by integrating the player order and generating a priority function from the temporal graph condition. Moreover the main result is presented: HR-games are decidable. Section 6 covers related work and the paper is concluded in Section 7.

2 HR-Grammars & Temporal Graph Conditions

In this section, we define hyperedge replacement grammars [7, 4] and introduce temporal graph conditions, i.e. graph conditions [8] equipped with temporal

operators. The basis are hypergraphs and hypergraph morphisms. Hypergraphs are a generalization of graphs where hyperedges are allowed to have a number of tentacles instead of one source and one target.

Assumption Let Σ be a finite, ranked alphabet and $N \subseteq \Sigma$ be a subset of *nonterminal* symbols where $rank\colon \Sigma \to \mathbb{N}$ [1] assigns to each symbol $a \in \Sigma$ its rank $rank(a)$.

Definition 1 (Hypergraph & Hypergraph Morphism) A *hypergraph* over Σ is a tuple $G = (V, E, att, lab, ext)$, consisting of a set of *nodes* V, a set of *hyperedges* E, an *attachment function* $att\colon E \to V^*$ [2], a labelling function $lab\colon E \to \Sigma$ and a (possibly empty) sequence $ext \in V^*$ of pairwise disjoint *external nodes*. For $e \in E$, we set $rank(e) = rank(lab(e))$ and require $|att(e)| = rank(e)$ [3]. A hypergraph is *finite* if the sets of nodes and edges are finite. We denote the empty graph as $\emptyset$. The components of a hypergraph G are denoted as $V_G, E_G, att_G, lab_G, ext_G$, respectively. Furthermore, $E_G^N = \{e | e \in E_G : lab(e) \in N\}$ denotes the set of *nonterminal hyperedges* of G, i.e. hyperedges labelled with nonterminal symbols.

Given two hypergraphs G, H, a *hypergraph morphism* $f\colon G \to H$ consists of functions $f_V\colon V_G \to V_H$ and $f_E\colon E_G \to E_H$, such that $f_V^*(att(e)) = att(f_E(e))$ [4] and $lab(e) = lab(f_E(e))$, i.e., morphisms preserve attachments and labels. A morphism is *injective*, *surjective* or an *isomorphism*, if f_V and f_E are, respectively, injective, surjective or both.

We use hypergraphs to define rules which replace a hyperedge by a hypergraph and connect them along the nodes connected to the hyperedge and the external nodes of the hypergraph.

Definition 2 (Hyperedge Replacement) A *hyperedge replacement rule* $r\colon X ::= R$ consists of a nonterminal label $X \in N$ and a hypergraph R over Σ where $rank(X) = |ext_R|$.

Let G, H be hypergraphs, $r : X ::= R$ a rule and $e \in E_G$ a nonterminal hyperedge with $rank(e) = |ext_R|$ and $lab(e) = X$. A *derivation* or *hyperedge replacement* from G to H by the rule r applied to e, written $G \Rightarrow_{r,e} H$ or $G \Rightarrow_r H$, is the substitution $H = G[R/e]$ of a hyperedge $e \in E_G$ by R, i.e.,

- $V_H = V_G \uplus (V_R \setminus [ext_R])$ [5]
- $E_H = (E_G \setminus \{e\}) \uplus E_R)$
- $att_H = att_G | (E_G \setminus \{e\}) \cup att_R; mod$ [6]

[1] $\mathbb{N}$ is the set of all natural numbers, including 0.

[2] V^* is the collection of all finite sequences over V, including the empty sequence ϵ.

[3] $|att(e)|$ denotes the length of a sequence $att(e)$.

[4] For a function $f\colon A \to B$, the free symbolwise extension $f^*\colon A^* \to B^*$ is defined by $f^*(a_1 ... a_n) = f(a_1) ... f(a_n)$.

[5] $\uplus$ denotes the disjoint union, $\setminus$ the difference of sets and $[ext_R]$ denotes the set of elements of ext_R.

[6] The function $f|S$ is the restriction of f to a set S. The symbol ; denotes forward composition of functions, i.e. $f; g(x) = f(g(x))$.

– $lab_H = (lab_G|(E_G \setminus \{e\})) \cup lab_R$
– $ext_H = ext_G$

where *mod* matches external nodes of R to the attachments of e, i.e. $mod = \mathrm{id}_{V_H} \cup \{[ext_R(1) \to att_G(e)(1), ..., ext_R(rank(e)) \to att_G(e)(rank(e))\}$.

Definition 3 (Hyperedge Replacement Grammar) A *hyperedge replacement (HR-)grammar* is a tuple $\mathfrak{G} = (G_0, \mathcal{R})$ where G_0 is the *start hypergraph* and $\mathcal{R}$ is a finite set of rules.

We extend hyperedge replacement grammars to *ordered* ones to obtain an analogue of left-most derivations [9]. For this purpose, we consider *ordered hypergraphs* in which the set of nonterminal hyperedges is ordered.

Definition 4 (Ordered Hyperedge Replacement Grammar) An *ordered* hyperedge replacement grammar is a hyperedge replacement grammar with an ordered start graph and a set of ordered rules.

An *ordered* hypergraph (G, o) consists of a hypergraph G together with a bijective function $o\colon E_G^N \to [1, n]$ [7] (or equivalently a sequence $o = e_1, e_2, ..., e_n$) where E_G^N denotes nonterminal hyperedges in G and $n = |E_G^N|$ is the number of elements in the set.

An *ordered hypergraph morphism* from (G, o_G) to (H, o_H) is a hypergraph morphism $f\colon G \to H$ that *respects the order*, i.e., for all $e, e' \in E_G$, $o_G(e) < o_G(e')$ implies $o_H(f(e)) < o_H(f(e'))$.

A rule $r : X ::= (R, o)$ is *ordered*, if (R, o) is an ordered hypergraph.

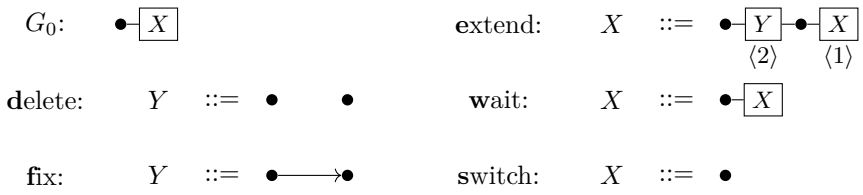
An *ordered derivation* from an ordered hypergraph (G, o_G) to an ordered hypergraph (H, o_H) by an ordered rule $r : X ::= (R, o_R)$ is a derivation from G to H in which the first nonterminal hyperedge in the order is replaced. The order of the resulting hypergraph H is obtained by substituting the hyperedge e_1 in the order o_G by the order o_R.

Notation For an ordered hypergraph G with $o = e_1 e_2 ... e_n$ and a natural number $i \leq n$, by the hyperedge e_i we refer to the edge at position i in o and $\mathtt{str}(G)$ produces the sequence of nonterminal labels in the graph in the reverse order of its hyperedges, i.e. $\mathtt{str}(G) = X_n X_{n-1} ... X_1$ where $X_i = lab(e_i)$.

In the following we write *graph*, *edge*, *rule* and *grammar* instead of ordered hypergraph, hyperedge, ordered rule and ordered hyperedge replacement grammar and G, H, R are always ordered hypergraphs.

Example 1 (Ordered Hyperedge Replacement Grammar) Consider the grammar $\mathfrak{G} = (G_0, \mathcal{R})$ with the start graph G_0 and rules $\mathcal{R} = \{$*extend*, *wait*, *switch*, *delete*, *fix*$\}$ as shown below. The position of a hyperedge in the respective order is marked by a label $\langle i \rangle$ with $i \in \mathbb{N}$. Unless otherwise specified, we assume both the attachment function and the order of hyperedges proceed left to right.

[7] $[1, n]$ denotes the set of natural numbers from 1 to n.



Definition 5 (Temporal Graph Condition) A *positive condition* is of the form $\exists(a : \emptyset \hookrightarrow C)$, where $a : \emptyset \hookrightarrow C$ is a morphism into a graph C, short $\exists(C)$. A positive condition c is *satisfied* by a graph G, written $G \models c$, if there exists an injective morphism q, such that for the morphism $i_G : \emptyset \hookrightarrow G$, we have $a; q = i_G$.

Temporal graph conditions are propositional LTL-formulas ([12]) where positive conditions are the *atomic conditions*. We use the usual abbreviations of *always* ($\Box$) and *eventually* ($\Diamond$) operators. For a temporal graph condition ϕ, the set of all atomic conditions is $At(\phi)$.

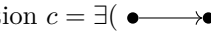
For the definition of the semantics of temporal graph conditions, we define labelled transition systems in the sense of [11] and *graph transition systems* analogously to [3].

A *labelled transition system* over some alphabet consists of a, potentially infinite, labelled graph with a designated start node and edges labelled over the alphabet. We refer to the nodes of a transition system as *states* and the edges as *transitions*. A *run* over a transition system is a sequence $\sigma = \sigma_0\sigma_1\sigma_2...$ of states along transitions of the transition system beginning with the start node that is either infinite or must end in a state without outgoing transitions.

Definition 6 (Graph Transition System) The *graph transition system* $\mathcal{T} = (T, s_0)$ of a grammar $\mathfrak{G} = (G_0, \mathcal{R})$ is a labelled transition system where T is a tree [8] with root node s_0 and $lab(s_0) = G_0$ where every state $s \in V_T$ is labelled with a graph, every transition $t \in E_T$ is labelled with a rule, and there exists a transition $s \xrightarrow{r} s'$ for states $s, s' \in E_T$ and $r \in \mathcal{R}$, if $lab(s) \Rightarrow_r lab(s')$.

Notation Where it is clear from context, we speak of *transition systems*, rather than labelled transition systems or graph transition systems.

Satisfaction of temporal graph conditions is defined over graph transition systems, where atomic conditions are evaluated over state labels.

Example 2 (Temporal Graph Condition) In the following, we use the temporal graph condition $\phi = \Box\neg c$ with the atomic condition $c = \exists($  $)$.

3 Parity Pushdown Games

In this section, we recall the definition of parity pushdown games [15]. We consider two player games, in which one player `sys` represents actions of a system

[8] A *tree* is a connected, cycle free graph with a designated root node.

and the other player `env` represents interference from the environment. Pushdown games are defined over pushdown processes, i.e. pushdown automata without acceptence features. We label the transitions of pushdown processes with rules and define transition systems of pushdown processes as counterparts to graph transition systems.

Definition 7 (Pushdown Process) Let $\mathcal{R}$ be a set of hyperedge replacement rules. A *pushdown process* over $\mathcal{R}$ is a tuple $\mathcal{P} = (Q, \Gamma, q_0, w_{init}, \delta)$, where Q is a finite set of *states*, Γ is a finite *stack alphabet*, q_0 is the start state, $w_{init} \in \Gamma^+$ [9] is the *initial stack content*, and $\delta \subseteq Q \times \Gamma \times \mathcal{R} \times Q \times \Gamma^*$ is the *transition relation*. A *configuration* is a pair (q, s), where $q \in Q$ is a state and $s \in \Gamma^+$ is the word called the *stack content*. The *initial configuration* is (q_0, w_{init}).

The *transition system* $\mathcal{T}(\mathcal{P}) = (T, s_0)$ of a pushdown process $\mathcal{P}$ is a labelled transition system where T is a tree with root s_0, $lab(s_0) = (q_0, w_{init})$ and there exists an edge $(q_i, s \cdot X, r) \xrightarrow{e} (q_j, s \cdot w)$ [10] with $lab(e) = (r, X, w)$, $q_i, q_j \in E_T$, if there is a corresponding transition $(q, r, X) \to (q', w)$ with $q, q' \in Q$ in the pushdown process $\mathcal{P}$.

A *pushdown transducer* $\mathcal{PT}$ is a pushdown process with additional input and output alphabets Σ_i, Σ_o and a partial output function $\lambda \colon Q \to \Sigma_o$.

Notation We order elements on the stack from left to right, such that the rightmost element is at the top of the stack.

Lemma 1 (Pushdown Store Languages are Regular [2])
Given a pushdown process with a set of *final states*, the *pushdown store language*, i.e. the language of words over Γ on the stack in a final state, is regular.

Given a pushdown process, every priority function induces a parity condition.

Definition 8 (Parity Condition) Given a pushdown process $\mathcal{P}$ with state set Q and a *priority function* $pri \colon Q \to \mathbb{N}$, the condition $Par(pri)$, with $Par(pri) \models \sigma$ if and only if the least priority $pri(\sigma_i)$ occurring infinitely often is even for a run $\sigma = \sigma_0\sigma_1\sigma_2...$ over $\mathcal{T}(\mathcal{P})$, is called *parity condition*.

A pushdown process along with a partition of its states and a priority function constitute a parity pushdown game.

Definition 9 (Parity Pushdown Game [15]) A *parity pushdown game* $\mathcal{G} = (\mathcal{P}, part, pri)$ consists of a pushdown process $\mathcal{P}$ along with a *partition function* $part \colon Q \to \{\texttt{sys}, \texttt{env}\}$ and a priority function $pri \colon Q \to \mathbb{N}$. A *run* over the transition system of $\mathcal{P}$ is *won* by player `sys` if it satisfies the parity condition and is won by player `env` otherwise. A *finite run* is *won* by player `sys` if it ends in a state of `env` and there is no applicable move for `env` and vice versa.

A strategy sets the next move for a player; to do so for the infinite state spaces of pushdown games we use a pushdown transducer.

[9] Γ^+ denotes the set of nonempty sequences over Γ.
[10] The symbol $\cdot$ denotes string concatenation.

Definition 10 (Strategy) A *strategy* for player `sys` is a pushdown transducer, which reads moves of `env` to produce the next move for `sys`. A *winning strategy* is a strategy such that a run $\sigma = \sigma_0\sigma_1\sigma_2...$ is won by `sys`, if for each σ_i where $part(\sigma_i) =$ `sys` the output function λ produces a move to the state σ_{i+1}.

Theorem 1 (Parity Pushdown Games are Decidable [15])
Finding a winning strategy for a parity pushdown game is decidable.

4 From HR-Grammar to Pushdown Process

In this section, we construct a pushdown process from an ordered hyperedge replacement grammar and a temporal graph condition.

4.1 Simulating a HR-Grammar by a Pushdown Process

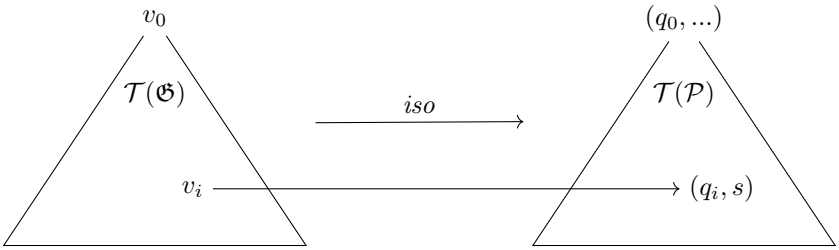
To construct a parity pushdown game from a grammar, we first need to construct a pushdown process. Any run over the transition system of the grammar has a corresponding run using the same rules over the transition system of the pushdown process. Additionally, the states of the pushdown process are sets of the atomic conditions of the winning condition which must be satisfied by the corresponding graphs generated by the grammar.

Definition 11 (Simulation) Let $\mathfrak{G}$ be a HR-grammar, ϕ be a temporal graph condition and $\mathcal{P}$ a pushdown process with sets of conditions over $At(\phi)$ as states. $\mathcal{P}$ *simulates* $\mathfrak{G}$ with respect to ϕ, if

(iso) there exists an isomorphism $iso\colon \mathcal{T}_u(\mathfrak{G}) \to \mathcal{T}_u(\mathcal{P})$ between their unlabelled transition systems,

(rules) for every transition e in $\mathcal{T}(\mathfrak{G})$, the transition $iso(e)$ in $\mathcal{T}(\mathcal{P})$ is labelled with the same rule as the edge e in addition to any stack changes, and

(sat) for every state v in $\mathcal{T}(\mathfrak{G})$ labelled with a graph G and the state $iso(v) = (q, s)$ in $\mathcal{T}(\mathcal{P})$, if $G \models c$ with $c \in At(\phi)$ then $\exists c' \in q : c' \Rightarrow c$ and if $c \in q$ then $G \models c$.



4.2 Partial Conditions

To be able to conclude which atomic conditions hold in a state of the pushdown process, we track which fragments of the condition are satisfied. We refer to these fragments as *subconditions*.

Definition 12 (Subcondition) For a condition $\exists(C)$, the condition $\exists(C')$ with $C' \subseteq C$ is called a *subcondition* of $\exists(C)$. The set of all subconditions of a condition c is denoted by $\texttt{Sub}(c)$. We write $c_s \sqsubseteq c$ for $c_s \in \texttt{Sub}(c)$.

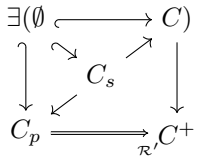
If a condition is satisfied, its subconditions are also satisfied.

Lemma 2 (Subconditions preserve Satisfaction) For a graph G and a condition $c = \exists(C)$, $G \models c$ implies $G \models c_s$ for all $c_s \in \texttt{Sub}(c)$.

Proof If $G \models c$, then there exists an injective morphism $q\colon C \hookrightarrow G$. Since c_s is a subcondition of c, $C_s \subseteq C$ and then there exists a morphism $a_s^+\colon C_s \to C$, such that $a_s^+; q$ is a morphism, implying $G \models c_s$. □

Subconditions with additional nonterminal hyperedges enable the application of rules to conditions, we call these *partial conditions*.

Definition 13 (Partial Condition) Let $c = \exists(C)$ be an atomic condition, and $c_s = \exists(C_s)$ a subcondition of c. A *partial condition* $c_p = \exists(C_p)$ *with respect to* c, is a condition with $C_s \subseteq C_p$, such that there exists a derivation $C_p \Rightarrow_{\mathcal{R}'} C^+$ to $C^+ \supseteq C$ by some set of rules $\mathcal{R}'$ where the number of hyperedges in $C_p \setminus C_s$ is at most the number of items in $C \setminus C_s$. For a condition c, $\texttt{Part}(c)$ is the set of all partial conditions of c.



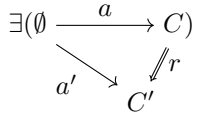
Lemma 3 (Finiteness of Partial Conditions) The set of partial conditions for a given condition c is finite.

Proof Follows directly from the finiteness of c and the limit on the additional hyperedges. □

4.3 Construction of the Pushdown Process

To create a pushdown process we construct states as sets of partial conditions with respect to the atomic conditions of ϕ and use the rules of $\mathfrak{G}$ to construct transitions between such states.

Let $c = \exists(C)$ be a partial condition, $r : X ::= R$ a rule, and $C \Rightarrow_{r,e} C'$ a derivation. Then $c' = \exists(C')$ is the derived condition. We write $c \Rightarrow_{r,e} c'$ or $c \Rightarrow_r c'$.



If a condition is satisfied by a graph, deriving a new condition will also allow the derivation of a graph that satisfies this condition. For ordered hyperedge replacement, this is only true if we replace the first nonterminal in the respective orders.

Lemma 4 (Satisfaction of Derived Conditions) Let G be an ordered hypergraph and $c = \exists(C)$ a partial condition, such that $G \models c$ by a morphism q. If $q(e_1) = e_1'$ where $e_1 \in C$ and $e_1' \in G$, a derivation $c \Rightarrow_{r,e_1} c'$ implies a derivation $G \Rightarrow_{r,e_1'} G'$, such that $G' \models c'$.

Proof Satisfaction of c implies the existence of a morphism $sat\colon C \hookrightarrow G$. Assuming r replaces an edge e in C, we derive G' by applying r to $sat(e)$. We can construct a morphism $sat'\colon C' \to G'$ by restricting sat to $C \setminus e$ and expanding it to include $R \subseteq C' \to R \subseteq G'$. The existence of sat' implies that $G' \models c'$. □

In general derived conditions will not be partial conditions. We split derived conditions by collecting all of their subconditions that are partial conditions with respect to atomic conditions. Since there are only finitely many partial conditions, we can use split to ensure that the construction generates a finite state set. The pushdown process is constructed by repeated application of rules to partial conditions, such that each rule application mirrors a rule application to a corresponding graph in the grammars transition system. Therefore, we also need to ensure that the conditions produced by split have no "gaps" with respect to the corresponding graphs.

Definition 14 (Split) The condition $c = \exists(C)$ *split* according to the atomic condition $c_t = \exists(C_t)$, results in the set

$$\begin{aligned} \mathit{Split}(c, c_t) = \{\exists(C_u) \mid \exists(C_p) \in \mathtt{Part}(c_t) \cap \mathtt{Sub}(c) : C_p \sqsubseteq C_u \sqsubseteq C, \\ \text{the order } o_{C_u} \text{ is the smallest uninterrupted subsequence of } o_C\} \end{aligned}$$

For an index i, $\mathit{Split}_i(c, c_t) = \{\exists(C_u) \in \mathit{Split}(c, c_t) | \text{edge } e_i \text{ in } o_C \text{ is } e_1 \text{ in } o_{C_u}\}$ and for sets S, S_t of conditions, $\mathit{Split}_i(S, S_t) = \bigcup_{c_t \in S_t} \bigcup_{c \in S} \mathit{Split}_i(c, c_t)$.

To determine which conditions correspond to modifiable parts of the graph, we need to differentiate between conditions that have a satisfying morphism that matches their first nonterminal to the first nonterminal of the graph and conditions that do not.

Definition 15 (Accessibly Satisfied) A partial condition c is *accessibly satisfied (a-satisfied) with respect to a morphism* $q\colon C \hookrightarrow G$, if $q(e_1) = e_1'$ where e_1 and e_1' are the first nonterminals in the order of C and G, respectively, and is *inaccessibly satisfied (i-satisfied) with respect to* q, if $q(e_1) \neq e_1'$. A partial condition is *a-satisfied* (*i-satisfied*) with respect to a graph G, if the condition is a-satisfied (i-satisfied) with respect to some morphism $C \hookrightarrow G$.

The pushdown process is constructed from states consisting of three sets, a-satisfied conditions A, i-satisfied conditions I, and conditions with multiple i-satisfying morphism M. Additional states A', I', M' are constructed by application of rules only to the a-satisfying conditions A. There are two cases:

Rule creates new nonterminal hyperedges In the first case, the rule creates new *nonterminal* hyperedges. The new a-satisfying conditions A' are the conditions derived from A which contain the first nonterminal of the right-hand side of the rule. Additionally, new i-satisfying conditions might be created which must be added to I', as well as potentially be added to M' in case they already exist in I. The new stack content w contains the newly created nonterminals. The i-th nonterminal is paired with conditions in I' that contain the nonterminal at $i-1$ in the order as their first. Additionally, the nonterminals are paired with conditions in M' whenever a new condition was added to M'. These are the sets α and μ, respectively.

Rule creates terminal items only In the second case, we only produce *terminal* items. This will lead to previously i-satisfying conditions becoming a-satisfying, thus they must be added to A' in addition to the conditions derived from A. These conditions were previously recorded in the set α on the stack. Conditions in α have to be removed from I' unless there are multiple i-satisfying morphisms, i.e. they also occur in M. Lastly, conditions may have to be removed from M if this has previously been recorded on the stack in the set μ.

Construction 1 (Transition) Let $q = (A, I, M)$ consisting of three sets of partial conditions be a state of a pushdown process, $(X, \alpha, \mu) \in \Gamma$ be a triple consisting of a nonterminal X and two sets of partial conditions α, μ and $r : X ::= R$ be a rule.

We construct a new state $q' = (A', I', M')$ and new stack content w from the set of derived conditions $D = \{c \mid a \in A : a \Rightarrow_r c\}$. There are two cases for filtering the conditions D:

(1) $\texttt{str}(R) = Y_n Y_{n-1} ... Y_1$
$A' = \mathit{Split}_1(D, \mathit{At}(\phi))$
$I' = I \cup \bigcup_{i=2}^{n} \mathit{Split}_i(D, \mathit{At}(\phi))$
$M' = M \cup \bigcup_{i=2}^{n} (I_i \cap \mathit{Split}_i(D, \mathit{At}(\phi)))$
$w = (Y_n, \alpha, \mu)(Y_{n-1}, \alpha_{n-1}, \mu_{n-1}) ... (Y_2, \alpha_2, \mu_2)(Y_1, \alpha_1, \mu_1)$
where $\alpha_i = \mathit{Split}_{i+1}(D, \mathit{At}(\phi))$, $\mu_i = I_{i+1} \cap \mathit{Split}_{i+1}(D, \mathit{At}(\phi)) \setminus M_{i+1}$, $I_j = I \cup \bigcup_{i=j+1}^{n} \mathit{Split}_i(D, \mathit{At}(\phi))$ and $M_j = M \cup \bigcup_{i=j+1}^{n} (I_i \cap \mathit{Split}_i(D, \mathit{At}(\phi)))$

(2) $\texttt{str}(R) = \epsilon$
$A' = \{C_p^1 | C_p \sqsubseteq C_p^1 \sqsubseteq C, \exists (C) \in D, \exists (C_p) \in \mathit{Split}(D, \mathit{At}(\phi))$
C_p^1 contains the first nonterminal in the order o_C,
$o_{C_p^1}$ is the smallest uninterrupted subsequence of $o_C\} \cup \alpha$
$I' = I \setminus (\alpha \setminus M)$
$M' = M \setminus \mu$
$w = \epsilon$

The resulting transition is $(q, (X, \alpha, \mu), r, w, q') \in \delta$.

Construction 2 (Pushdown Process $\mathcal{P}(\mathfrak{G}, \phi)$) Given a hyperedge replacement grammar $\mathfrak{G} = (G_0, \mathcal{R})$ and a temporal graph condition ϕ, $\mathcal{P}(\mathfrak{G}, \phi) = (Q, \Gamma, q_0, w_{init}, \delta)$ is the pushdown process where the set of states Q, the stack

alphabet Γ, initial state q_0, initial stack content $w_{init} \in \Gamma^+$ and the transition relation δ are constructed as follows:

States are triples of sets of partial conditions. Let $q_{aux} = (\emptyset, \emptyset, \emptyset)$ be a helper state that is not part of Q and a let $(X, \emptyset, \emptyset) \in \Gamma$. The start state q_0 and initial stack content w_{init} are constructed from q_{aux} according to Construction 1, except that we assume $D = \{\exists(G_0)\}$. The generated state is q_0 and $w_{init} = \bot \cdot w$.

All other states and transitions are constructed by application of Construction 1 for as long as new states or transitions are generated.

By Lemma 1, we can determine the pushdown store language for each state by treating that state as the final state of the pushdown process, including the triples (X, α, μ) that may occupy the top of the stack, which allows us to construct additional states and transitions.

Notation States $q = (A, I, M)$ of pushdown processes are represented as shown on the left below. The upper part of the node contains the set A, while the lower part contains I and M. Conditions c in the lower part are marked c^+ when they are present in both I and M. We abbreviate the partial conditions and triples $(X, \alpha, \mu) \in \Gamma$ as shown on the tables below.

$\{c_{yx}, c_x\}$
$\{c_{yy}, c_y^+\}$

name	condition
c	$\exists(\bullet \longrightarrow \bullet \longrightarrow \bullet)$
c_x	$\exists(\bullet - \boxed{X})$
c_y	$\exists(\bullet - \boxed{Y} - \bullet)$
c_{yx}	$\exists(\bullet - \boxed{Y} - \bullet - \boxed{X})$
c_{yy}	$\exists(\bullet - \boxed{Y} - \bullet - \boxed{Y} - \bullet)$
$c_{y\rightarrow}$	$\exists(\bullet - \boxed{Y} - \bullet \longrightarrow \bullet)$

name	stack symbol
$X_{\emptyset,\emptyset}$	$(X, \emptyset, \emptyset)$
$X_{y,\emptyset}$	$(X, \{c_y\}, \emptyset)$
$X_{yy,\emptyset}$	$(X, \{c_{yy}, c_y\}, \emptyset)$
$X_{yy,y}$	$(X, \{c_{yy}, c_y\}, \{c_y\})$
$X_{yy,yy}$	$(X, \{c_{yy}, c_y\}, \{c_{yy}\})$

To make use of these pushdown processes for solving HR-games, we first need to show that they simulate the grammars they are constructed from.

Theorem 2 (Simulation) The pushdown process $\mathcal{P}(\mathfrak{G}, \phi)$ is finite and simulates $\mathfrak{G}$ with respect to ϕ.

Proof By induction over a run γ over $\mathcal{T}(\mathfrak{G})$ and Lemmas 2 and 4. $\mathcal{P}(\mathfrak{G}, \phi)$ is finite: By Lemma 3 the number of partial conditions is finite and their uninterrupted versions are finite, since we have a finite set of rules. □

Example 3 (Pushdown Process with Partial Conditions) We take the grammar from Example 1 and the temporal graph condition from Example 2 to construct the pushdown process partially shown in Fig. 2.

5 Construction of Hyperedge Replacement Games

The pushdown process from the previous section must be modified to generate a partition of its states and priority function to form a game. To generate

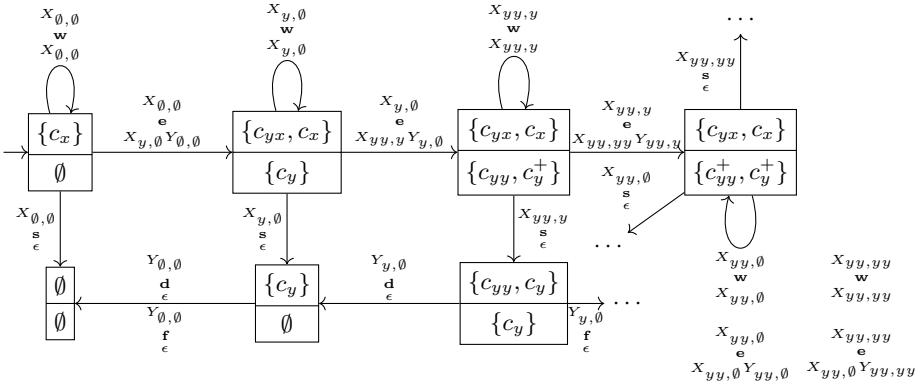


Fig. 2: Pushdown Process with Partial Conditions

a partition, we need to modify the pushdown process $\mathcal{P}(\mathfrak{G}, \phi)$ such that any state assigned to a player by the partition only has outgoing transitions that correspond to rules that belong to that player.

Example 4 (Player Order & Division of Rules) We specify the order in which players may make their moves with a transition system with states labelled over $\{\texttt{sys}, \texttt{env}\}$. In addition we set $\mathcal{R}(\texttt{sys}) = \{\mathit{extend}, \mathit{delete}\}$ and $\mathcal{R}(\texttt{env}) = \{\mathit{wait}, \mathit{switch}, \mathit{fix}\}$ for the rules introduced in Example 1.

The following transition system `ord` specifies alternating turns for the players `sys` and `env`:

Construction 3 (Restriction) Let $\mathcal{P} = (Q, \Gamma, q_0, \bot, \delta)$ be a pushdown process over $\mathcal{R}$, $\texttt{ord} = (T, s_0)$ a labelled transition system over $\{\texttt{sys}, \texttt{env}\}$.

The *restriction* of $\mathcal{P}$ by `ord` is constructed by synchronizing $\mathcal{P}$ and `ord` such that transitions $(q_i, (X, \alpha, \mu), r, w, q_j) \in \delta_{\mathcal{P}}$ are synchronized with edges in `ord` that start at a state labelled with `sys`, if $r \in \mathcal{R}(\texttt{sys})$ (analogously for `env`). The labels $\texttt{sys}, \texttt{env}$ of states of `ord` induce a partition $\mathit{part} \colon Q \to \{\texttt{sys}, \texttt{env}\}$ over the states of of the modified pushdown process.

We are left to integrate the temporal graph condition ϕ, which we translate into a priority function over the pushdown process.

Note The result of Walukiewicz [15] assumes a pushdown process with an empty stack at the start. We can easily modify the pushdown process $\mathcal{P}(\mathfrak{G}, \phi)$ by reintroducing the auxiliary state q_{aux} used during construction as the new start state connected to q_0 by a transition that creates the initial stack. We set $\mathit{part}(q_{aux}) = \texttt{env}$.

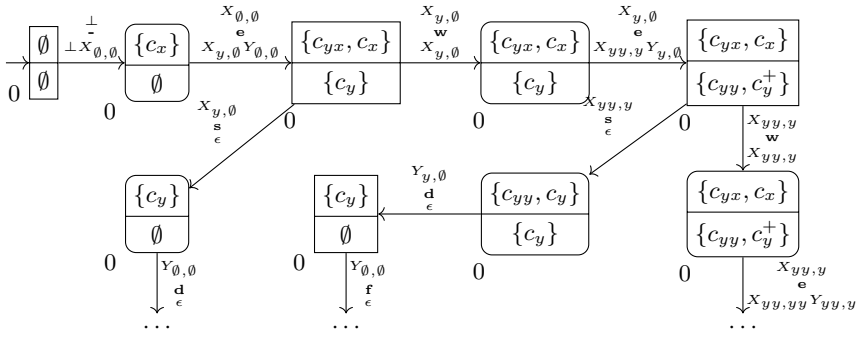


Fig. 3: Final Pushdown Process

Lemma 5 (Temporal Graph Condition to Parity Condition) There is a transformation from a pushdown process $\mathcal{P}$ and a temporal graph condition ϕ to a priority function *pri* over $\mathcal{P}$, such that an infinite run σ over $\mathcal{T}(\mathcal{P})$ satisfies the parity condition $Par(pri)$ iff the run satisfies ϕ.

Construction 4 (*pri*) Given a pushdown process $\mathcal{P}$ and a temporal graph condition ϕ, we modify $\mathcal{P}$ and construct a priority function *pri* over $\mathcal{P}$.

Convert ϕ to a Büchi-automaton $\mathcal{A}_\phi$ [14] and interpret $\mathcal{A}_\phi$ as a parity automaton where all accepting states have priority 0 and all other states have priority 1. To integrate the priorities into $\mathcal{P}$ we synchronize $\mathcal{P}$ with $\mathcal{A}_\phi$ by treating a transition $(q_i, (X, \alpha, \mu), r, w, q_j) \in \delta_\mathcal{P}$ with $q_j = (A, I, M)$ as if it is labelled c, if $\exists c' \in A \cup I : c \sqsubseteq c'$ and $\neg c$ otherwise for all $c \in At(\phi)$. We set $pri(q_{aux}) = 0$ and determine *pri* for other states by the component from $\mathcal{A}_\phi$.

Proof [14] establishes this equality for ϕ and $\mathcal{A}_\phi$. The construction moves from a sequence of transitions in $\mathcal{A}_\phi$ to a sequences of states in $\mathcal{T}(\mathcal{P})$, the additional state q_{aux} has no influence on the satisfaction of $Par(pri)$. □

Example 5 (Final Pushdown Process) The final pushdown process for the grammar from Example 1, the temporal graph condition from Example 2 and the player order from Example 4 is shown in Fig. 3. The partition function *part* is indicated by rectangles for `env` and rounded corners for `sys`. Additionally, the priority function *pri* is shown by a label at the bottom left corner of each state.

With these results, we define hyperedge replacement games.

Definition 16 (HR-Game) Let $\mathfrak{G}$ be an ordered hyperedge replacement grammar, `ord` a transition system with states labelled over $\{\texttt{sys}, \texttt{env}\}$ and ϕ a temporal graph condition. A *hyperedge replacement (HR) game* $\mathcal{G} = (\mathcal{P}, part, pri)$ consists of the pushdown process $\mathcal{P}(\mathfrak{G}, \phi)$ derived by Construction 2 and modified according to `ord` by Construction 3 and ϕ by Construction 4. *part* and *pri* are the partition function and the priority function induced by the latter two constructions.

Theorem 3 (HR-Games) Finding a winning strategy for HR-games is decidable.

Proof Follows directly from the Theorems 1 and 2 and Lemma 5. □

For the example game we have constructed throughout the paper, we cannot construct a winning strategy for `sys`. While there is no state in the restricted pushdown process that satisfies c, there is a strategy for the environment player `env` that forces the game to end in a state in which `sys` has no available moves.

6 Related Work

The combination of graph transformation and games has also been considered by Kaiser [10], referred to as *structure rewriting games*. In contrast to our approach rules are applied to all possible matches simultaneously. In addition, the approach uses a restricted class of graph transformation, called *separated handle rewriting*. The combination of restricted rewriting rules and simultaneous application to all matches also leads to decidable games.

Graph Transformation Games have independently been considered by Alabdullatif and Heckel [1], in the context of negotiation games. In contrast to our graph transformation games, players in the negotiation game play with the goal of minimizing costs rather than a binary win/loss. Consequently the solutions to these games are not winning strategies for either player, but an equilibrium for both players.

7 Conclusion and Future Work

In this paper we have proposed an approach to the verification of the correctness of graph grammars under adverse conditions, i.e. under the influence of an adverse environment. A hyperedge replacement game, consisting of an ordered hyperedge replacement grammar, a labelled transition system defining a player order and a temporal graph condition is introduced as a model for this purpose. We establish decidability of these games by showing that they can be reduced to parity pushdown games.

The main contributions of the paper are (1) a new notion of graph transformation games for modeling correctness under adverse conditions and (2) the identification of a decidable fragment of graph transformation games via a reduction to parity pushdown games.

It should be possible to extend the construction in such a way that winning conditions can be temporal conditions over any nested condition, since it should still suffice to consider a collection of finite subgraphs of a state. The main restriction of the translation to a pushdown process is the necessity for *ordered* hypergraph grammars. Dropping this restriction should still allow the construction of a Petri net, rather than a pushdown process, that similarly simulates a grammar. Unlike for pushdown games, however, we do not have a corresponding solution for finding winning strategies for transition systems generated by Petri nets.

Acknowledgements We would like to thank Annegret Habel, Reiko Heckel, Berthold Hoffmann and Mark Minas for helpful feedback on earlier versions of this paper.

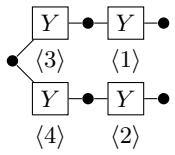
References

1. Alabdullatif, M., Heckel, R.: Graph transformation games for negotiating features. In: Graph Computation Models (GCM 2016) (2016)
2. Autebert, J.M., Berstel, J., Boasson, L.: Context-free languages and pushdown-automata. In: Handbook of Formal Languages, vol. 1: Word Language Grammar, pp. 111–172. Berlin (1997)
3. Baldan, P., Corradini, A., König, B., Lluch Lafuente, A.: A temporal graph logic for verification of graph transformation systems. In: Proc. of WADT '06. LNCS, vol. 4409, pp. 1–20 (2007)
4. Drewes, F., Habel, A., Kreowski, H.J.: Hyperedge replacement graph grammars. In: Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1, pp. 95–162. World Scientific (1997)
5. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamental theory of typed attributed graph transformation based on adhesive HLR categories. Fundamenta Informaticae **74(1)**, 31–61 (2006)
6. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research, LNCS, vol. 2500. Springer (2002)
7. Habel, A.: Hyperedge Replacement: Grammars and Languages, LNCS, vol. 643. Springer (1992)
8. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. MSCS **19**, 245–296 (2009)
9. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation - (2. ed.). Addison-Wesley series in computer science, Addison-Wesley-Longman (2001)
10. Kaiser, L.: Synthesis for structure rewriting systems. In: Mathematical Foundations of Computer Science 2009, 34th International Symposium, MFCS 2009. pp. 415–426 (2009)
11. Keller, R.M.: Formal verification of parallel programs. Communications of the ACM **19**(7), 371–384 (Jul 1976)
12. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA. pp. 46–57. IEEE Computer Society (1977)
13. Poskitt, C.M., Plump, D.: Verifying partial correctness of graph programs. Electronic Communications of the EASST (2013)
14. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Proceedings of the First Symposium on Logic in Computer Science. pp. 322–331. IEEE Computer Society (1986)
15. Walukiewicz, I.: Pushdown processes: Games and model-checking. Information and Computation **164**(2), 234–263 (2001)

A Additional Examples and Proofs

Example 6 (Interrupted Order of Hyperedges in Derived Partial Conditions) The rule double below, combined with the condition $c = \exists(\bullet\longrightarrow\bullet\longrightarrow\bullet)$ from Example 2, could lead to an 'interruption' in the order of hyperedges. We apply the rule to the partial condition $c' = \exists(\bullet\text{–}\boxed{X})$, the resulting condition c'' is *not* a partial condition, since there are more additional hyperedges than items missing from c. There is a partial condition $\exists(\bullet\text{–}\boxed{Y}\text{–}\bullet\text{–}\boxed{Y}\text{–}\bullet)$ that is a subcondition of c'', but compared to c'' we 'skip' the Y-hyperedge at position 2. Since we intend to construct our pushdown process by applying rules to partial conditions, we need to prevent this situation.

double: X ::=



Example 7 (Split) Let $c' = \exists(\bullet\text{–}\boxed{Y}\text{–}\bullet\text{–}\boxed{X})$ be a partial condition of the atomic condition $c = \exists(\bullet\longrightarrow\bullet\longrightarrow\bullet)$. A condition $c'' = \exists(\bullet\text{–}\boxed{Y}\text{–}\bullet\text{–}\boxed{Y}\text{–}\bullet\ \boxed{X})$ can be derived by applying the rule extend from Example 1 to c'. c'' however, is *not* a partial condition, since there are more hyperedges than there are items missing from c. It has subconditions $c_s \sqsubseteq c''$ which are. We collect these subconditions which are also partial conditions, such that $\mathit{Split}(c', c) = \{\exists(\bullet\text{–}\boxed{Y}\text{–}\bullet\text{–}\boxed{X}), \exists(\bullet\text{–}\boxed{Y}\text{–}\bullet), \exists(\bullet\text{–}\boxed{X})\}$.

Example 8 (Example State Change) We show an example of a state change according to Construction 1 in Fig. 4. We start with the state $q = (A = \{\exists(\bullet\text{–}\boxed{X})\}, I = \emptyset, M = \emptyset)$, assume a stack content $(X, \alpha = \emptyset, \mu = \emptyset)$ and apply the rule extend from Example 1. First, we generate the set $D = \{\exists(\bullet\text{–}\boxed{Y}\text{–}\bullet\text{–}\boxed{X})\}$, by applying extend to the a-satisfying condition $\exists(\bullet\text{–}\boxed{X}) \in A$ of q. The new state q' is constructed by splitting the conditions of D. A new i-satisfying condition $\exists(\bullet\text{–}\boxed{Y}\text{–}\bullet)$ is generated, since it is a partial condition of the target condition $c = \exists(\bullet\longrightarrow\bullet\longrightarrow\bullet)$ from Example 2. This condition is also present in the new stack content w, indicating in which step it should be moved from i-satisfying to a-satisfying conditions.

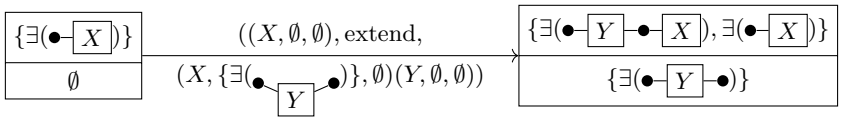


Fig. 4: State change with partial conditions

Proof (of Theorem 2) To use Lemma 4 to show (sat), we will need to show that for a state $\gamma_i \in \mathcal{T}(\mathfrak{G})$ with $lab(\gamma_i) = G_i$ and a corresponding configuration $\rho_i = (q, S_i)$ with $iso(\gamma_i) = \rho_i$ and $q = (A, I, M)$, the conditions in A are a-satisfying, the conditions in I are i-satisfying and there is more than one i-satisfying morphism for the conditions $M \subseteq I$. By construction A, I, M might contain (a) a condition c' that is isomorphic to an atomic condition c or (b) condition c' with $c \sqsubseteq c'$, in either of those cases $c' \Rightarrow c$.

We show (iso), (rules) and (sat) by induction over the states of a run $\gamma = \gamma_0\gamma_1...$ over $\mathcal{T}(\mathfrak{G})$.

Start:

(iso):
For $\gamma_0 \in \mathcal{T}(\mathfrak{G})$ with $lab(\gamma_0) = G_0$ there is a corresponding configuration $\rho_0 \in \mathcal{T}(\mathcal{P}(\mathfrak{G}, \phi))$, which is the start state and initial stack (q_0, S_0) created in Construction 2. We set $iso(\gamma_0) = \rho_0$.

(rules):
The nonterminal at the tail of the initial stack S_0 for $\mathcal{P}(\mathfrak{G}, \phi)$ matches $X_1 \in G_0$ by Construction 2.

(sat), "$\Rightarrow$":
A: By Construction 2, $q_0 = (A_0, I_0, M_0)$ such that $A_0 = Split_1(D)$ where $D = \{\exists(G_0)\}$. $Split_1(D)$ consists of those subconditions that preserve $X_1 \in G_0$, thus for each $c = \exists(\emptyset \to C) \in Split_1(D)$ there exists a morphism $sat\colon C \to G_0$ (due to Lemma 2) with $sat(Y_1) = X_1$, such that $Y_1 \in C$. Therefore, all $c \in A_0$ are a-satisfying with respect to G_0.

I: The conditions in I_0 are constructed from D by uninterrupted subcondition. By Lemma 2, these conditions are satisfied, since D is trivially satisfied. Additionally, the conditions in I are i-satisfying by Construction 2.

M: By Construction 2, the conditions in M_0 are conditions which occur more than once as subconditions of $\exists(\emptyset \to G_0)$, thus there is more than one i-satisfying morphism for these conditions.

(sat), "$\Leftarrow$":
By Construction 2 q_0 contains all subconditions of $\exists(\emptyset \to G_0)$ which are partial conditions. Those are all partial conditions satisfied by G_0.

Step:

Assume that for γ_i with $lab(\gamma_i) = G_i$ we have $\rho_i = (n, S_i)$ with $q = (A, I, M)$, such that $iso(\gamma_i) = \rho_i$, that $X_1 \in G_i$ matches the nonterminal at the tail of S_i. Additionally, the conditions $c = \exists(\emptyset \to C) \in A$ are a-satisfying with respect to G_i, conditions in I are i-satisfying with respect to G_i and conditions in $M \subseteq I$ have more than one i-satisfying morphism. Finally, assume that for all $c \in \mathcal{E}$ iff $G_i \models c$ then $c \in A \cup I$ or $\exists c' \in A : c \sqsubseteq c'$.

(iso):
There exists an edge $\gamma_i \xrightarrow{e} \gamma_{i+1}$ in $\mathcal{T}(\mathfrak{G})$ with $lab(e) = r$ and $lab(\gamma_{i+1}) = G_{i+1}$ where $r : X ::= R$ is a rule applicable in G_i. By Construction 2 q has a transition for a rule r applicable to the tail X of its stack language, which is equivalent to $X_1 \in G_i$ by assumption. This transition implies the existence of an edge

$\rho_i \xrightarrow{e'} \rho_{i+1}$ in $\mathcal{T}(\mathcal{P}(\mathfrak{G}, \phi))$ with $lab(e') = (X, r, w)$, such that $\rho_{i+1} = (q', S_{i+1})$, we set $iso(e) = e'$ and $iso(\gamma_{i+1}) = \rho_{i+1}$.

(rules):

In both cases, r is applied, the first nonterminal in the resulting graph G_{i+1} must therefore match the tail of S_{i+1}.

(sat), "⇒":

The conditions A', I', M' of q' vary based on the rule $r : X ::= R$, as follows:

1. $\texttt{str}(R) = Y_1 Y_2 ... Y_n$
   A': The a-satisfying conditions $c' \in A'$ consist of subconditions $Split_1(D)$. By Lemma 4 and Lemma 2, these conditions have satisfying morphisms sat'. By assumption conditions in A are a-satisfying, then the conditions in D must be a-satisfying, since we can easily construct a suitable satisfying morphism for the derived conditions in D. Then conditions c' must be a-satisfying with respect to sat', since $Split_1(D)$ preserves the first nonterminal of conditions in D.
   I': The i-satisfying conditions in I' of q' consists of conditions in I and new conditions generated by application of r. Conditions in I, which were already satisfied in ρ_i by assumption, remain satisfied: None of the conditions in I are a-satisfying with respect to G_i, hence the targets of their satisfying morphisms may not be replaced. As a consequence conditions in I remain satisfied. These conditions remain i-satisfying, since the application of r creates additional nonterminals. The new conditions in $Split_2(D) \cup Split_3(D) \cup ... \cup Split_n(D)$, are satisfied due to Lemma 4 and Lemma 2 and are i-satisfying by construction.
   M': Similarly, the conditions in M' consists of conditions in M and new conditions generated by application of r. Conditions in M have more than one i-satisfying morphism by assumption. The new conditions in M' have more than one i-satisfying morphism by Construction 2, as they are only added to M' if there is already an instance in I.
2. $\texttt{str}(R) = \epsilon$
   If no new nonterminals are generated by the application of r, we need to consider the transition $\gamma_o \xrightarrow{e_o} \gamma_{o+1}$ where $lab(\gamma_o) = G_o$ with $lab(e_o) = (X_o, r_o, w_o)$ that created $X_2 \in G_i$, where $r_o : X_o ::= R_o$ and $\texttt{str}(R_o) = Z_1 Z_2 ... Z_m$ [11]. With w_o added to the stack, it must contain the triple (X, α, μ) removed from the stack in the current transition. Assume that $X_1 \in G_i$ corresponds to some $X_k \in G_o$.

$$\texttt{str}(R_o) = \begin{array}{|c|c|c|c|c|c|} \hline Z_m & Z_{m-1} & \cdots & Z_k & \cdots & Z_0 \\ \hline \alpha & \alpha_{m-1} & \cdots & \alpha_k & \cdots & \alpha_0 \\ \hline \mu & \mu_{m-1} & \cdots & \mu_k & \cdots & \mu_0 \\ \hline \end{array} \Big\} w_o$$

[11] We need not consider the case $\texttt{str}(R) = \epsilon$, which never adds to the stack.

We have already considered the conditions $c'' = \exists(\emptyset \to C'') \in \alpha_k$ in case 1 above, hence we know that there are satisfying morphisms sat'' for these conditions.

A': The a-satisfying conditions in A' of q' consist of two sets:

(a) Conditions derived from A using r:
Conditions $c' \in \{unint(c' \cup X_1, c) | c' \in Split(c) \wedge X_1 \in C$ for $c = \exists(\emptyset \to C) \in D\}$ are subconditions of some $c \in D$. By Lemma 4 conditions in D are satisfied and by Lemma 2 subconditions preserve satisfaction. Additionally, the conditions contain $X_1 \in C$ by construction and are therefore a-satisfying.

(b) Previously i-satisfying conditions α:
The set α was generated at an earlier transition e_o (see above) and added to the nonterminal X on the stack. At γ_{o+1} we can further assume that for c'', we have $sat''(Y_1) = X_{m-k}$ with $Y_1 \in C''$ and $X_{m-k} \in G_{o+1}$. Later rules before r cannot replace any of the nonterminals of conditions up to α_k, since they are not a-satisfying.
After application of r to G_i, we move the conditions c'' to A', the morphisms sat'' are still satisfying morphisms. As a consequence of the replacement of $Z_{k+1}...Z_n$, we now have $sat''(Y_1) = X_1$ with $Y_1 \in c''$ and $X_1 \in G_{i+1}$.

I': The i-satisfying conditions I' of q' consist of conditions from I, which are i-satisfying in G_i by assumption. Consider again an earlier transition e_o. For conditions in I to become a-satisfying in G_{i+1}, $Z_k...Z_m$ must have already been replaced. Then the conditions that become a-satisfying in G_{i+1} are exactly those stored in α_k during application of r_o (as shown for α above). These conditions have to be removed from I to generate I', unless there is more than one i-satisfying morphism for them which is true for the conditions in M by assumption. Hence conditions in $I' = I \setminus (\alpha \setminus M)$ are i-satisfying.

M': The conditions in M' of q' consist of conditions from M, which have multiple i-satisfying morphisms in G_i by assumption. Consider again an earlier transition e_o.For a condition in M to no longer have several i-satisfying morphisms in G_{i+1}, the replacement of X must have made the second instance of an i-satisfying condition a-satisfying. If this is the case, Construction 1 must have added the condition to μ_k during application of r_o. These conditions are removed from M to form M'. Hence conditions in $M' = M \setminus \mu$ have multiple i-satisfying morphisms.

(sat), "$\Leftarrow$":

We show (sat) for G_{i+1} and q' by contradiction:

Assume there is some partial condition c_p generated by case (2) of Construction 1 such that $G_{i+1} \models c_p$ but neither $c_p \in A' \cup I'$ nor $\exists c' \in A' \cup I' : c_p \sqsubseteq c'$. By Construction 1 new conditions in A' and I' are introduced by deriving conditions from A using r and by use of *Split*, i.e. subconditions of the derived conditions. To be satisfied in G_{i+1} the condition c_p must either be the result of applying r or already be satisfied by G_i and therefore $c_p \in A \cup I$ or $\exists c' \in Acc \cup I : c_p \sqsubseteq c'$ by assumption.

If $c_p \in A$ or $\exists c' \in A : c_p \sqsubseteq c'$, then for c_p to also be satisfied in G_{i+1} the derivation of c'_p from c_p by r must result in $c_p \sqsubseteq c'_p$ and since c_p is a partial condition we will have $c_p \in A' \cup I'$. If $c_p \in I$ is must either persist in I' or be moved to A'. In either case we have a contradiction to our initial assumption.

Otherwise the satisfaction of c_p in G_{i+1} must be the result of the application of r (as in the first case above), which is a contradiction to our assumption again. □