Bachelorarbeit

Deciding FO over Languages of Hypergraphs

vorgelegt durch:
Max Goertzt

Gutachter:
Priv.-Doz. Dr. Thomas Noll
Prof. Dr. Ir. Joost-Pieter Katoen

Erklaerung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstaendig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die woertlich oder sinngemae aus veroeffentlichten und nicht veroeffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht.

Aachen, den 16. Juli 2012

(Max Goertz)

Contents

1 Introduction

The description of sets of words (called languages) is a well known field of research. However a less popular extension of this concept is formed by languages of relational structures like hypergraphs. But still there are many applications working on these structures. Most of these cases require ways to describe languages of structures. For hypergraphs one possibility is given by hyperedge replacement grammars. In analogy to context free string grammars they form a system of nonterminal hyperedges(symbols), hypergraphs(words) and production rules that specify how nonterminal hyperedges can be replace in order to derive different hypergraphs.

One example application that puts hyperedge replacement grammars to use tis the verification tool Juggrnaut[6]. This framework abstracts heap configurations as hypergraphs. Furthermore most of the use cases also involve the testing for certain properties. For structures like hypergraphs many ways to define them exist. A straight forward approach is provided by the so called graph conditions[4]. These conditions are defined as a hyperedge replacement grammar themselves. Checking a graph condition results in testing equality of languages. Another suitable form for defining characteristics of relational structures is given by *LTL*[5]. This logic can be used to define existence of certain paths inside the structures. There are some more ways do define properties for relational structures. However this thesis puts the focus on two possibilities for characterizing hypergraphs namely the first oder logic *FO* and the monadic second order logic *MSO*. Both include quantification over vertices and predication about hyperedges. *MSO* extends *FO* by the quantification over subsets of vertices.

Deciding properties for a single hypergraph is mostly trivial. Most of the applications require statements about whole languages of hypergraphs though. This holds as well for Juggrnaut. Checking properties for languages of structures is

not decidable in general however the decidability of *MSO*-formulas over hypergraphs was already proven by B. Courcelle[3]. But, and this is the motivation of this thesis, to our knowledge no applicable algorithm to decide this problem exists. During this thesis we will develop such an algorithm and thereby provide a different, constructive proof of the decidability.

At the beginning of the thesis we will start with some preliminary definitions. We will give an definition of hypergraphs that is slightly different to most common definitions[1] but is better suited for our purpose. Afterwards we give according definitions for hyperedge replacement and hyperedge replacement grammars. Finally we introduce first order logic as a first possibility of defining hypergraph properties.

Then we start to develop an algorithm to decide *FO* over hypergraphs by discussing the base case of single terminal hypergraphs. This leads then to an generalization to nonterminal hypergraphs and thereby to languages of hypergraphs.

In the end we introduce *MSO* as second more powerful logic and explain how the algorithm can be altered in order apply to *MSO* formulas

The central idea of our approach is to modify hyperedge replacement grammars in a way that allows us to specify properties for the single nonterminal symbols that enable us to make a statement about nonterminal hypergraphs by simply looking at its nonterminal hyperedges.

2 Preliminaries

2.1 Tuples

Tuples are vectors of values and will be marked with a bar: $\bar{s} \in S^*$.
For a set S and $n \in \mathbb{N}$, S^n is the set of n-tuples of S-elements. The set S^* is defined as $\bigcup_{n\in\mathbb{N}} S^n$. For a tuple $\bar{s} \in S^*$, $|\bar{s}|$ denotes the length of s and for $i \in \mathbb{N}$ $\bar{s}[i]$ represents the ith element of $\bar{s}$. Given two tuples $\bar{s} \in S^*$ and $\bar{m} \in M^*$, the product $\bar{s} \bullet \bar{m} \in (S \cup M)^*$ denotes the tuple where $|\bar{s} \bullet \bar{m}| = |\bar{s}| + |\bar{m}|$, $(\bar{s} \bullet \bar{m})[i] = \bar{s}[i]$ for $1 \leq i \leq |\bar{s}|$ and $(\bar{s} \bullet \bar{m})[i] = \bar{m}[i - |\bar{s}|]$ for $|\bar{s}| < i \leq |\bar{s} \bullet \bar{m}|$.
The set S^0 only contains the empty tuple ϵ for any set S.

Given a function $f : S \to M$ we allow the use of an implicit extension $f : S^* \to M^*$. For $\bar{s} \in S^n$ this function is defined so that $f(\bar{s}) \in M^n$ with $f(\bar{s})[i] = f(\bar{s}[i]), 1 \leq i \leq n$. This means that applying a function to a tuple is equivalent to applying it to every element and then forming a new tuple.

Given a natural number $n \in \mathbb{N}$, $\underline{n}$ denotes the set of all natural numbers that are less or equal to n: $\{m \mid m \in \mathbb{N}, m \leq n\}$.

2.2 Hypergraphs

Hypergraphs are an generalization to regular graphs. The difference is that edges are not represented as a binary relation. This is because the a so called hyperedge does not necessarily connect two vertices. Instead hyperedges may connect an arbitrary number of vertices. Even hyperedges with no connected vertex are allowed. To introduce the concept of direction to these hyperedges the exists an order over the attached vertices. A common graph with binary directed edges would for example have edges where attachment number one represents the source and attachment number two the destination of the edge. One vertex may af course occur multiple times in this listing. This is e.g. needed to realize self loops on vertices.

In Addition the hypergraphs use here contain so called external vertices. These are just a subset of the vertices of the hypergraph. In the section about hyperedge replacement grammars their existence will be motivated.

Within this thesis we will discuss how to scan hypergraphs for certain properties. We will even check these properties for possibly infinite sets of hypergraphs. How to define a property and a infinite set of hypergraphs will be explained in the section about *FO* and hyperedge replacement grammars respectively.

Definition 1 (Ranked Alphabet)**.** *A ranked alphabet Σ is a set of symbols with an associated ranking function rk. This means for every symbol $a \in \Sigma$ we have a rank $rk(a) \in \mathbb{N}_0$. These ranked alphabets are used to label hyperedges of matching rank where the rank of a hyperedge is the count of attached vertices.*

Definition 2 (Labeled Hypergraph)**.** *A labeled hypergraph over a ranked alphabet Σ is a tuple (n, E, att, lab, ext) where $n \in \mathbb{N}$ is the number of vertices and E the set of Edges. Vertices are represented by their numbers. The set V of vertices therefore is $\underline{n}$. Every hyperedge $e \in E$ is labeled by the function $lab : E \to \Sigma$. The function $att : E \to V^*$ maps a hyperedge to the connected vertices. The equation $|att(e)| = rk(lab(e))$ has to hold for every $e \in E$. Therefore we denote $rk(e) = rk(lab(e))$. The number ext denotes the number of external vertices and might be zero. If ext is non zero the set of external vertices is $\underline{ext}$ otherwise it's empty.*

The set of hypergraphs over a ranked alphabet Σ is denoted by HG_Σ.

Example 1 (Hypergraph). In this figure an example hypergraph is printed. It has 3 nodes. Two of them (printed bold) are external. Furthermore there are 3 hyperedges labeled with L_1 and L_2. The ranked alphabet therefore is a superset of $\{L_1, L_2\}$ with $rk(L_1) = 3$ and $rk(L_2) = 2$. The whole hypergraph is described by:

$$\begin{aligned} G &= (3, \{e_1, e_2, e_3\}, att, lab, 2) with \\ att &= \{e_1 \to (1,2,3), e_2 \to (3,2), e_3 \to (1,3)\}, \\ lab &= \{e_1 \to L_1, e_2 \to L_2, e_3 \to L_2\} \end{aligned}$$

Like in this example we will always display vertices as circles and hyperedges as squares. The numbers inside the vertices are their id. As the hyperedges don't

have any specified identifier the displayed square only contains the associated label. The lines that connect vertices with hyperedges are used to visualize the attachment function. Every hyperedge e has $rk(e)$ of these connections. A vertex v being connected to a hyperedge e by a i labeled joint implies $att(e)[i] = v$.

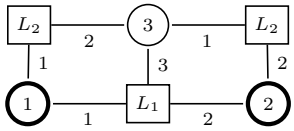


Figure 1: Example hypergraph

2.3 Hyperedge Replacement Grammars

Like string grammars that are used to represent languages of strings, hyperedge replacement grammars generate languages of hypergraphs. A language of hypergraph is a possibly infinite set of hypergraphs. In analogy to string grammars hyperedge replacement grammars consist of a set of derivation rules that have a nonterminal symbol on the left hand and a hypergraph on the right hand side.

Definition 3 (Hyperedge Replacement Grammar)**.** *For a ranked alphabet Σ we define a hyperedge replacement grammar as the tuple (N, T, P, S) where N denotes the set of nonterminal and T the set of terminal symbols with $N \cup T = \Sigma$ and $N \cap T = \emptyset$. The set P contains production rules of the form $X \to G$ where $X \in \Sigma$ and $G \in HG_\Sigma$. We require that $rk(X) = ext_G$. The last entry of the grammar is the starting symbol $S \in N$. We require the rank of this starting symbol to be zero.*

The set of all hyperedge replacement grammars over Σ is denoted by HRG_Σ.

A hyperedge e is called terminal if $lab(e) \in T$ and a hypergraph is terminal if it contains only terminal hyperedges. In the following terminal symbols will be represented by lowercase and non-terminal symbols by uppercase letters.

Given a hyperedge replacement grammar one can derive a possibly infinite set of hypergraphs by starting with the hypergraph containing only the S denoted hyperedge and then successively replacing nonterminal hyperedges in the current graph according to the production rules. These replacements are explained in detail below.

Example 2 (Hyperedge replacement grammar). The Figure below shows an example of a hyperedge replacement grammar with two rules. The grammars is defined as $G = (N, T, PS)$ with

$$\begin{aligned} N &= \{S\} \\ T &= \{p, q\} \end{aligned}$$

There are two production rules. The first one maps X onto a hypergraph with 3 vertices and 2 hyperedges (1 terminal, 1 nonterminal). The second rule is a terminal one. The rank of the right hand sides is (as required) equal. The rank $rk(X)$ is 2.

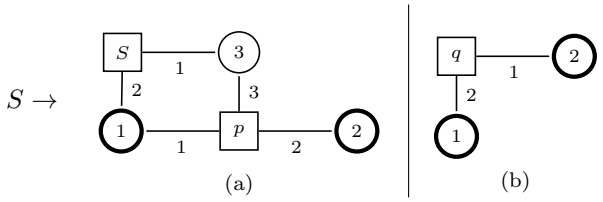


Figure 2: Example hyperedge replacement grammar

Definition 4 (Hyperedge Replacement)**.** *Let $K, H \in HG_\Sigma$ be two hypergraphs and $e \in E_K$ a non-terminal hyperedge in I with $rk(e) = ext_H$. W.l.o.g. we assume that the edges of the two hypergraphs are distinct i.e. $E_K \cap E_H = \emptyset$.*

When replacing hyperedges we embed one hypergraphs into another one. The embedded hypergraph H replaces the hyperedge e in the embedding hypergraph K by merging its external vertices with the ones attached to e. The external vertex with id 1 is merged with the first vertex attached to e and so on. As one vertex may be attached multiple times multiple external vertices can be combined into one. The internal vertices of H remain without any connection to H. We refer to the embedding hypergraph K as the context of K.

The hypergraph obtained by replacing e by H in K is called $K[H/e]$:

$$K[H/e] = (n, E, att, lab, ext)$$

The vertices of this new hypergraph are obtained by adding the vertices of the hypergraph H to those of K. As the external vertices of H are merged with nodes of K these nodes do not have to be added. Therefore the resulting vertex count is:

$$n = n_K + n_H - ext_H$$

The external vertices are those of G:

$$ext = ext_K$$

Except for the replaced hyperedge every hyperedge from K and H exists in the resulting hypergraph as well:

$$E = (E_K \setminus \{e\}) \cup E_H$$

As the vertex ids in $K[H/e]$ have to be in $\underline{ext}$ and unique we have to rename some vertices. This is achieved by keeping vertex ids for the vertices of K while the vertices of H are given subsequent numbers. As mentioned above external

vertices from H do not need additional ids as they are merged with existing vertices. Formally this means:

$$newId(v) = \begin{cases} att(e)[v], & v \leq ext_H \\ n_K + v - ext_H, & otherwise \end{cases}$$

The attachments of the hyperedges are accordingly:

$$att(x) = \begin{cases} att_K(x), & x \in E_K \\ newId(att_H(x)), & x \in E_H \end{cases}$$

One important property of hyperedge replacement is that terminal hyperedges can never be deleted by replacing hyperedges. During every step one nonterminal but never a terminal hyperedge is removed. Therefore a terminal hyperedge that exists at some point will always remain no matter how many hyperedge replacements are executed.

Example 3 (Hyperedge replacement). Figure 3 shows an example of hyperedge replacement. The X-labeled hyperedge in K is replaced by H. The result is depicted below. The internal vertex 3 of H is appended to the vertices of K and get the new id 4. As vertex 2 is attached two times (indicated by the $1, 3$) to the replaced hyperedge. The external vertices 1 and 3 of H are merged.

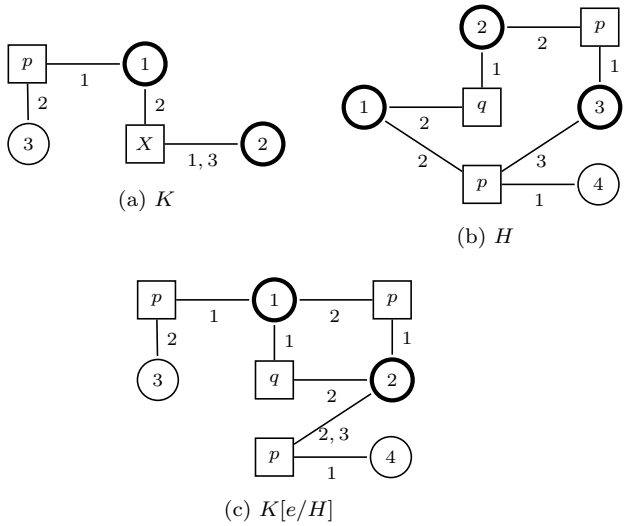


Figure 3: Example hyperedge replacement

Definition 5 (*HRG* Derivation)**.** *Let $G \in HRG_\Sigma$, be a hyperedge replacement grammar and $H, H' \in HG_\Sigma$ hypergraphs over the same alphabet. For a nonterminal hyperedge $e \in E_H$ and a production rule $p = X \to K$, H derives H' by*

p iff $H' = H[K/e]$. *This derivation relation is visualized by* $H \overset{p,e}{\Longrightarrow} H'$. *If this production rule is part of the hyperedge replacement grammar* ($p \in =_G$) *we write* $H \overset{G}{\Longrightarrow} H'$. *The the reflexive-transitive closure of* $\overset{G}{\Longrightarrow}$ *is denoted by* $\overset{G}{\Longrightarrow}^*$.

If a hypergraph K *is derivable from* H *by application of* n *production rules of* G *we say "the derivation of* K *from* H *has lengthn" or* $H \overset{G}{\Longrightarrow}^n K$.

As an abbreviation we define the relation $\overset{G}{\Longrightarrow}^*$ *for nonterminal symbols too. Given a nonterminal symbol* X *we write* $X \overset{G}{\Longrightarrow}^* H'$ *iff a rule* $X \to H \in P_G$ *exists with* $H \overset{G}{\Longrightarrow}^* H'$

Similar to context free string grammars the relation $\overset{G}{\Longrightarrow}^*$ intuitively defines a the set of derivable hypergraphs:

$$derivable(H, G) = \left\{ H' \mid H \overset{G}{\Longrightarrow}^* H' \right\}$$

Example 4 (*HRG* Derivation). We now take the hyperedge replacement grammar from Example 2 and look at some possible derivations. As specified we start with the hypergraph containing only the S-labeled hyperedge. We then apply rule (a) which leads to hypergraph on the right hand side of this rule. By applying the same rule another time we add a terminal hyperedge and the vertex 4 to the hypergraph. Finally by applying rule (b) we derive a terminal hypergraph.

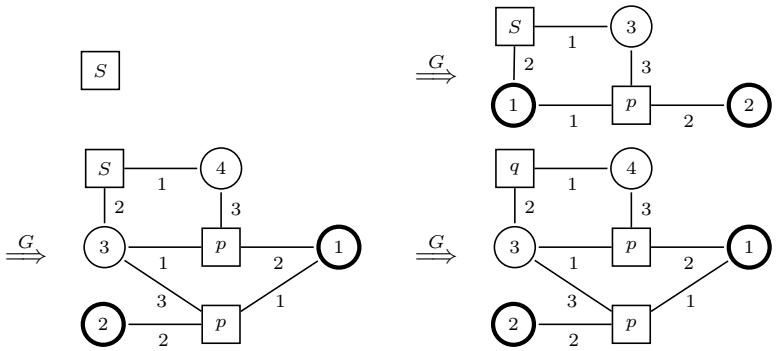


Figure 4: Example *HRG* derivation

Definition 6 (Decomposition Lemma)**.** *The decomposition lemma states that for any given hypergraph*$H = (n, E, att, lab, ext)$, *hyperedge replacement grammar* $G = (N, T, P, S)$ *and* $e \in E$ *with* $lab(e) = X \in N$ *the derivation*

$$X \overset{G}{\Longrightarrow}^* K$$

implies that

$$H \overset{G}{\Longrightarrow}^* H[e/K]$$

This lemma is very intuitive. More information is given at[2].

Definition 7 (Language of a *HRG*)**.** *Given a hyperedge replacement grammar $G \in HRG_\Sigma$, the language $\mathcal{L}(G)$ is defined by*

$$\mathcal{L}(G) = \left\{ H' \in HG_\Sigma \mid H_S \overset{G}{\Longrightarrow}^* H', H' \text{ is terminal} \right\}$$

where H_S is the hypergraph that only contains the starting hyperedge S:

$$H_S = (0, E_S = \{e_S\}, att_S = \emptyset, lab_S = \{e_S \to S\}, 0)$$

This language is a set containing every terminal hypergraph that's derivable by the given hyperedge replacement grammar.

Equally to the context free string grammars we're interested in the terminal hypergraphs we can generate. If cyclic dependencies between rules in G occur the language may become infinite.These cases are of course the interesting and useful ones.

2.4 Restriction of *HRG*s

To simplify some of the upcoming definitions we demand two restrictions to the the *HRG*s.

We first want the grammars to be start separated. This means the starting symbol does not occur on any right hand side: For all $N \to H \in P$ and every $e \in E_H$ we require $lab(e) \neq S$. This for can be easily achieved by adding a new starting symbol with only one rule generating the hypergraph containing the old starting symbol. This does not alter the language of the hypergraph.

The second restriction is that no vertex may be attached to one nonterminal hyperedge multiple times. More formally for all $N \to H \in P$ and every $e \in E_H$ with $ext(e)[i] \neq ext(e)[j]$ for $i \neq j$. This normal form does however not change the expressiveness of the *HRG*s. For every possible *HRG* G there is another *HRG* G' with $L(G) = L(G')$ and G' has the requested form.

To generate this new grammar one can add additional nonterminal symbols that represent the cases where one or more external vertices are identical. Given a nonterminal symbol $e(lab(e) = N)$ and a composition $\mathcal{C} \subset \mathcal{P}(\underline{rnk(e)})$ with $C \cap C' = \emptyset$ for $C, C' \in \mathcal{C}$ and $C \neq C'$, we add the nonterminal symbol N' to the alphabet with $rnk(N') = |\mathcal{C}|$. We consider the composition to be linearly ordered ($\mathcal{C} = \{C_1, C_2, ...\}$). For every rule $N \to H$ with $H = (n, E, att, lab, ext)$ we add a new one $N' \to H'$, $H' = (n', E, att', lab, ext')$ with

$$\begin{aligned}
ext' &= |\mathcal{C}| \\
n' &= n + ext' - ext \\
att'(e)[i] &= j && \text{if } att(e)[i] \in C_j \\
att'(e)[i] &= att(e)[i] + ext' - ext && \text{if } att(e)[i] > ext
\end{aligned}$$

This means we just merge the external vertices on the right hand sides according to the composition. Now if somewhere a hyperedge e with $lab(e) = N$ and

$\left\{\{j \mid att(e)[j] = att(e)[i]\} \mid i \in \underline{rnk(e)}\right\} = \mathcal{C}$ occurs, we replace this occurence with a N'-labeled hyperedge which has no vertex attached more than once.

2.5 First Order Logic

As a first way to describe properties of hypergraphs we introduce the language of first order logic formulas.

Definition 8 (*FO* over hypergraphs)**.** *FO formulas are simply constructed from negation, conjunction, disjunction, universal quantification and existential quantification over atomic formulas namely true, false and predicates. We define sets $FO(T, X)$ of formulas depending on a set T of terminal symbols and a set of variables X. W.l.o.g. We consider the variables to be ordered ($X = \{x_1, ..., x_m\}$).*

We first define the simplest atomic formulas.

$$true, false \in FO(T, X) \text{ for any } T, X$$

The second class of atomic formulas are the predicates. Predicates are used to reason about existence of terminal hyperedges.

$$p(\bar{x}) \in FO(T, X) \text{ if } p \in T, \bar{x} \in X^* \text{ and } rk(p) = |\bar{x}|$$

This formula states the existence of a p-labeled hyperedge attached to the vertices specified by $\bar{x}$.

Now we will discuss the logical operations (negation, conjunction and disjunction)

$$\begin{aligned}
\neg\varphi &\in FO(T, X) \quad \text{iff } \varphi \in FO(T, X) \\
\varphi \wedge \psi &\in FO(T, X) \quad \text{iff } \varphi \in FO(T, Y), \psi \in FO(T, Z) \text{ and } X \supseteq Y \cup Z \\
\varphi \vee \psi &\in FO(T, X) \quad \text{iff } \varphi \in FO(T, Y), \psi \in FO(T, Z) \text{ and } X \supseteq Y \cup Z
\end{aligned}$$

Finally we include the quantifiers

$$\begin{aligned}
\forall x\, \varphi &\in FO(T, X) \quad \text{iff } \varphi \in FO(T, Y) \text{ and } X \supseteq Y \setminus \{x\} \\
\exists x\, \varphi &\in FO(T, X) \quad \text{iff } \varphi \in FO(T, Y) \text{ and } X \supseteq Y \setminus \{x\}
\end{aligned}$$

Usually the definition of *FO* also includes an atomic formula $x_i = x_j$ to test two variables for equality. However for the sake of simplicity we decided to add *eq*-labeled self looping hyperedges to each of the vertices. Then the check of equality become an check whether this edge exists $x_i = x_j \rightarrow eq(x_i, x_j)$. For the sake of readability we will not print these hyperedges when displaying hypergraph.

Definition 9 (Free variables)**.** *Given a formula $\varphi \in FO(T, X)$, the minimal set $free(\varphi)$ such that $\varphi \in FO(T, free(\varphi))$ is called the set of free variables.*

Variables that occur in φ but are associated to a quantifier are called bound variables.

The formulas that we will examine are supposed to be saturated(i.e. $free(\varphi) =$). We will use FO instead of $FO(T, \emptyset)$ where T is the set of nonterminals of the current hyperedge replacement grammar.

Definition 10 (Variable assignment)**.** *Given a formula $\varphi \in FO(T, X)$ we define a variable assignment as a tuple $\bar{a} \in V^{|X|}$ where V is a set of vertices. Application of a variable assignment to a formula is denoted by $\varphi(\bar{a})$. We substitute a vector of variables by the vector of assigned vertices by using the substitution operator $\times$. For a variable vector $\bar{x} \in X^n$ it is defined as:*

$$\bar{x} \times \bar{a} = \bar{v} \text{ where } \bar{v} \in V^n \text{ and } \bar{v}[i] = \bar{a}[\bar{x}[i]]$$

We allow to write φ instead of $\varphi(\epsilon)$ if $\varphi \in FO(T, \emptyset)$.

Definition 11 (Semantics of *FO*-formulas)**.** *Given a terminal hypergraph $H \in HG_\Sigma$ and a formula $\varphi \in FO(T, \emptyset)$ we define a set of rules to determine whether H satisfies φ ($H \vDash \varphi$):*

1. $H \vDash \psi(\bar{a})$ *if* $\psi = true$.
2. $H \vDash \psi(\bar{a})$ *if* $\psi = p(\bar{x})$ *and a hyperedge* $e \in E_H$ *exists with* $att(e) = \bar{x} \times \bar{a}$.
3. $H \vDash \psi(\bar{a})$ *if* $\psi = \neg\omega$ *and* $H \nvDash \omega(\bar{a})$.
4. $H \vDash \psi(\bar{a})$ *if* $\psi = \omega \wedge \gamma$ *with* $H \vDash \omega(\bar{a})$ *and* $H \vDash \gamma(\bar{a})$.
5. $H \vDash \psi(\bar{a})$ *if* $\psi = \omega \vee \gamma$ *with* $H \vDash \omega(\bar{a})$ *or* $H \vDash \gamma(\bar{a})$.
6. $H \vDash \psi(\bar{a})$ *if* $\psi = \forall x\, \omega$ *and* $H \vDash \omega(\bar{a} \bullet v)$ *for every* $v \in \underline{n_H}$.
7. $H \vDash \psi(\bar{a})$ *if* $\psi = \exists x\, \omega$ *and* $H \vDash \omega(\bar{a} \bullet v)$ *for at least one* $v \in \underline{n_H}$.

For *FO*-formulas without quantifiers and predicates we consider a function $evaluate : FO \to \{true, false\}$ to be given.

Example 5 (*FO*-formulas). We will now look at some example formulas.

$$\varphi_{direction} = \forall x\ \forall y; \neg(p(x, y)) \vee p(y, x)$$

This first formula can be used to describe undirected graphs. It states that for every edge another edge in the opposite direction exists.

$$\varphi_{circle} = \exists x\ \exists y\ \exists z\ p(x, y) \wedge p(y, z) \wedge p(z, x)$$

This formula describes a directed circle of length 3. However this definition allows the circle to include loops. The formula describing circles of arbitrary length can not be defined in *FO*.

$$\varphi_{three} = \exists x\ \exists y\ \exists z\ \neg(eq(x, y)) \wedge \neg(eq(x, y)) \wedge \neg(eq(x, y))$$

Finally this formula checks whether a hypergraph contains at least three different vertices.

Some more properties that can not be defined in FO are

1. connectivity
2. finiteness
3. colorability
4. ...

All these properties can only be formulated if an upper bound for the number of vertices is known in advance. In general patterns that concern arbitrary many vertices can not be defined in FO.

Definition 12 (Semantics of FO-formulas for nonterminal hypergraphs)**.** *As the main focus in this thesis is on defining properties for languages of hypergraphs we want to extend the $\vDash$ relation to nonterminal hypergraphs. The idea is to simply look if every derivable terminal hypergraph satisfies the formula. This however brings up one problem. This decision can not be understood as a yes/no question. For some hypergraphs H and formulas φ we will get two derivations $H \overset{G}{\Longrightarrow}{}^{*} K_1, K_2$ with $K_1 \vDash \varphi$ but $K_2 \nvDash \varphi$. Therefore FO becomes a three-valued logic for languages of hypergraphs.*

Let G be a hyperedge replacement grammar and $\varphi \in FO$. When deciding φ over the language of hypergraphs we write

$$
\begin{aligned}
G \vDash \varphi \quad & \text{if } H \vDash \varphi \text{ for every } H \in \mathcal{L}(G) \\
G \nvDash \varphi \quad & \text{if } H \nvDash \varphi \text{ for every } H \in \mathcal{L}(G) \\
G \approx \varphi \quad & \text{if } H_1 \vDash \varphi \text{ for a } H_1 \in \mathcal{L}(G) \text{ and } H_2 \nvDash \varphi \text{ for a } H_2 \in \mathcal{L}(G)
\end{aligned}
$$

Definition 13 (Prenex normal form)**.** *From now on every FO-formula φ is considered to be in prenex normal form. Formulas in this form can be split in two parts φ_Q and $\varphi_{\bar{Q}}$ where φ_Q has the form $Q_1 x_1 Q_2 x_2 ...$ where $Q_i \in \{\forall, \exists\}$ and no quantifiers occur in $\varphi_{\bar{Q}}$. However this is no restriction as every FO-formula can be transfered into an equivalent one in prenex normal form by applying these few rules:*

$$
\text{For } Q \in \{\forall, \exists\} :
$$

$$
\begin{aligned}
\varphi \wedge Qx\,\psi \quad &\equiv Qx\,(\varphi \wedge \psi) \\
\varphi \vee Qx\,\psi \quad &\equiv Qx\,(\varphi \vee \psi) \\
\neg(\forall x\,\varphi) \quad &\equiv \exists x\,(\neg\varphi) \\
\neg(\exists x\,\varphi) \quad &\equiv \forall x\,(\neg\varphi)
\end{aligned}
$$

We refer to $q = qr(\varphi)$ as the quantifier rank of φ.

The quantifier free rest $\varphi_{\bar{Q}}$ of the formula only consists of the predicates and true/false formulas combined by negation, con- and disjunction. We consider the predicates to be ordered and the set of occuring predicates to be

$$Pred(\varphi) = \left\{ \psi_i \mid \psi_i \text{ occurs in } \varphi, \psi = p(\bar{x}) \text{ for } p \in N, \bar{x} \in X^{rk(p)} \right\}$$

The single predicates can be referenced by $pred(\varphi, i) = \psi_i$. The number of different predicates is called the predicate rank of φ and denoted by $pr(\varphi) = |Pred(\varphi)|$.

3 Evaluation for terminal hypergraphs

To start we will take a look at a strategy for deciding *FO*-formulas on a terminal hypergraph and generalize this approach to nonterminal hypergraphs.

The idea of the chosen approach is to first evaluate the quantifier free part of the formula for every possible variable assignment. Then we stepwise add quantifiers to the formula starting with the innermost one. The number of free variables in the resulting formula is reduced by one. Let this now bound variable be x. In order to evaluate this new formula we again take the every possible assignment for the remaining free variables. Now we consecutively assign every possible vertex to the variable x and take a look at the result previously evaluated for this assignment. If the added quantifier is an existential quantifier and assigning we just need the formula to evaluate to *true* for one of the x-values. For universal quantifiers we accordingly need one falsifying evaluation. An example for this evaluation strategy is depicted below:

$\varphi = \forall x_1 \, \exists x_2 \, \psi(x_1, x_2)$

$\psi(x_1, x_2)$		
$x_1 \backslash x_2$	1	2
1	*true*	*false*
2	*true*	*true*

$\exists x_2\, \psi(x_1, x_2)$	
x_1	
1	*true*
2	*true*

$\forall x_1 \, \exists x_2 \, \psi(x_1, x_2)$
true

Figure 5: Evaluaton idea

As described below first only the quantifier free part of the formula is considered. The assignments are represented by the different cells of the table. This formula (which doesn't need to be specified here) evaluates to *true* for the assignments $(x_1, x_2) = (1, 1), (2, 1), (2, 2)$ and falsifies only for $(x_1, x_2) = (1, 2)$. For the second table the $\exists x_2$ is added to the formula. The table dimension is reduced by one as x_2 is no longer a free variable. To calculate the table entries for

$x_1 = 1$ we have to look at the combinations with possible x_2 values. These combinations are $(1,1)$ and $(1,2)$. For $x_1 = 2$ the evaluations for $(2,1)$ and $(2,2)$ are of interest. As we have an existential quantifier only one of these entries needs to be *true* in order to get a positive result. In both cases we have at least one *true* entry. Therefore we get a *true* in both table cells. Finally we also add the $\forall x_1$. As all variables are bound now we just have one cell left. It contains the evaluation of the whole formula φ over the hypergraph. We again look at the entries for every variation of x_1 values in the previous table (1 and 2). This time we have an universal quantifier but as both entries are *true* we again evaluate to *true*. In this example the property specified byφ would apply to the given hypergraph.

3.1 Evaluation tables

Now we will formalize the tables used in the example above.

Definition 14 (Evaluation table)**.** *To evaluate the given formula φ on a given hypergraph $H = (n, E, att, lab, ext)$ we create a so called evaluation table $t \in ETable(\varphi, n)$ with $t = (V, entry)$. This t is a d-dimensional table containing one entry for every combination of d vertices. The entries are vectors (tuples) of m true/false values. Note that $d = qr(\varphi)$ and $m = pr(\varphi)$ are given by the formula as quantifier count and number of occurring predicates respectively. The number n is the number of identified vertices while the set $V = \underline{n}$. The formal representation of this table is realized by the function*

$$entry : V^d \to \{true, false\}^m$$

One remarkable change compared to the above example table is type of the entries. Instead of one true/false value we now store one value for every predicate. The idea is to store for every one of these predicates if it's satisfied (the corresponding hyperedge exists) or not. Knowing the formula φ this information also yields the value for the whole formula. This representation contains some extra information that will later be necessary though. During the next chapter about evaluation for nonterminal hypergraphs this necessity will be explained.

3.2 Table generation

Now that all the required definitions are made the generation of evaluation tables using the previously presented strategy is simple. Let H be a hypergraph. First we define how to get the single values for the different predicates. This is done by simply applying the semantics of FO-formulas. For the ith ($i \in \underline{m}$) predicate and the variable assignment $\bar{a} \in V^d$ we get:

$$evalPred(i, \bar{a}) = \begin{cases} true & \text{if } H \vDash \psi(\bar{a}) \text{ where } \psi = pred(\varphi, i) \\ false & \text{otherwise} \end{cases}$$

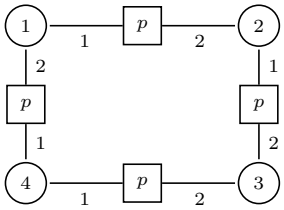
Now we only have to combine the values and put them into the table $table(H, \varphi) = (V, entry)$ with

$$\begin{aligned} V &= \underline{n_H} \\ entry(\bar{a}) &= evalPred(1, \bar{a}) \bullet ... \bullet evalPred(m, \bar{a}) \end{aligned}$$

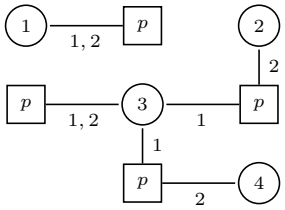
Example 6 (Table generation). In Figure 6 we can see two examples for tables generated for a specific hypergraph and formula. To save some space we abbreviated *true* as 1 and *false* as 0.

In the first table $table(K, \varphi_K)$ we only have one predicate $p(x, y)$. Therefore we get a *true* entry in every cell (x, y) for which a hyperedge between x and y exists. There are four hyperedges in K and accordingly the table contains four *true* entries ($(1, 2)$, $(2, 3)$, $(4, 3)$, $(4, 1)$).

The second formula φ_H contains 2 predicates. Therefore the tables consists of 2-valued vectors (written as comma separated values). The first predicate is *true* if a hyperedge between x and y exists and the second if a loop exist on y. There are two loops ($y = 1, 3$) and two other hyperedges ($(3, 2)$, $(3, 4)$). The values of the tables are set accordingly. However as the second predicate is independent from x every second value is *true* in the first and third row.



(a) K



(b) H

$\varphi_K = \forall x \, \exists y \, p(x, y)$

$y \backslash x$	1	2	3	4
1	0	0	0	1
2	1	0	0	0
3	0	1	0	1
4	0	0	0	0

$\varphi_H = \exists x \, \forall y \, p(x, y) \vee p(y, y)$

$y \backslash x$	1	2	3	4
1	0, 1	0, 1	0, 1	0, 1
2	0, 0	0, 0	1, 0	0, 0
3	0, 1	0, 0	0, 1	0, 0
4	0, 0	0, 0	1, 0	0, 0

Figure 6: Example for table generation

3.3 Interpretation of evaluation tables

Given a hypergraph H, a *FO*-formula φ and an associated evaluation table $t = (V, entry)$ we now want to formalize the algorithm deciding whether $H \vDash \varphi$ holds. Our example already gave an idea how to decide this.

First we want to condense the stored vectors to one single *true*/*false* value. To do so we can easily replace the predicates in the formula by the corresponding stored value. The corresponding formula can immediately be evaluated. We therefore define a function that carries the replacement of predicates out:

$$replacePred(\psi, \bar{b}) = \begin{cases} \bar{b}[i] & \text{if } \psi = pred(\varphi, i) \\ replacePred(\rho) \wedge replacePred(\omega) & \text{if } \psi = \rho \wedge \omega \\ replacePred(\rho) \vee replacePred(\omega) & \text{if } \psi = \rho \vee \omega \\ \neg replacePred(\rho) & \text{if } \psi = \neg\rho \end{cases}$$

The result of this function are formulas consisting of *true*, *false* and the logical operators $(\wedge, \vee, \neg)$. Using the *evaluation*-function directly yields the desired *true*/*false* value.

$$evalTable(t) = et(t, \epsilon)$$

The function $et(t, \bar{a}) : (ETable(\varphi, n), V^*) \to \{true, false\}$ where $V = \underline{n} \cup A$ is defined recursively. The base case is $\bar{a} \in V^{qr(\varphi)}$. Then

$$et(t, \bar{a}) = evaluate(replacePred(\varphi_{\bar{Q}}, entry(\bar{a}))$$

For the remaining cases we have to look at the quantifiers of φ. To evaluate $et(t, \bar{a})$ we set $i = |\bar{a}| + 1$. Then we look at the ith quantifier of φ.

If $Q_i = \forall$:

$$et(t, \bar{x}) = \begin{cases} false & \text{if } et(t, \bar{x} \bullet v) = false \text{ for a } v \in V \\ true & \text{otherwise} \end{cases}$$

If $Q_i = \exists$:

$$et(t, \bar{x}) = \begin{cases} true & \text{if } et(t, \bar{x} \bullet v) = true \text{ for a } v \in V \\ false & \text{otherwise} \end{cases}$$

This fully defines the evaluation function for evaluation tables.

Example 7 (Table interpretation). The tables in Figure 7 present the process of table evaluation. Starting with the initial table the following tables display the values of $et(\bar{a}, t)$ with $\bar{a} \in V^n$ for $n = 2, 1, 0$.

The first step is to insert the values for the predicates into the formula $\varphi_{\bar{Q}}$ at the corresponding position and then evaluate it. However in the first example the formula consists only of this single predicate. The second case is slightly more interesting. The assignment $(x = 1, y = 1)$ for example yields the formula *false*$\vee$ *true*, the assignment $(x = 3, y = 2)$ yields *true* $\vee$ *false* and the assignment$(x = 2, y = 1)$ yields *false* $\vee$ *false*. The first two cases evaluate to *true* while the third one produces a *false* entry.

When reducing the dimension of the tables by one we have to look at the

according quantifier. If the quantifier is universal we have to look for a *false* entry otherwise for a *true* one. Like this we finally get one single value telling us whether the formula is satisfied in the hypergraph.

In the first example the result is *false*. The second one evaluates to *true*. Both evaluations are correct.

$y\backslash x$	1	2	3	4
1	0	0	0	1
2	1	0	0	0
3	0	1	0	1
4	0	0	0	0

$\varphi''_K = p(x,y)$

$y\backslash x$	1	2	3	4
1	0, 1	0, 1	0, 1	0, 1
2	0, 0	0, 0	1, 0	0, 0
3	0, 1	0, 0	0, 1	0, 0
4	0, 0	0, 0	1, 0	0, 0

$\varphi''_H = p(x,y) \vee p(y,y)$

$y\backslash x$	1	2	3	4
1	0	0	0	1
2	1	0	0	0
3	0	1	0	1
4	0	0	0	0

$\varphi'_K = \exists y\ p(x,y)$

$y\backslash x$	1	2	3	4
1	1	1	1	1
2	0	0	1	0
3	1	0	1	0
4	0	0	1	0

$\varphi'_H = \forall y\ p(x,y) \vee p(y,y)$

x	1	2	3	4
	1	1	0	1

$\varphi_K = \forall x\ \exists y\ p(x,y)$

false

x	1	2	3	4
	0	0	1	0

$\varphi_H = \exists x\ \forall y\ p(x,y) \vee p(y,y)$

true

Figure 7: Example for table generation

3.4 Correctness

The correctness of evaluation tables follows directly from the semantics of *FO*. Let H be and hypergraph, φ be a *FO*-formula and $t = table(H, \varphi) = (n, A, entry)$. Given an assignment $\bar{a} \in V^q r(\varphi)$ the following obviously holds:

$$entry(\bar{a})[i] = true \text{ iff } H \vDash \psi(\bar{a}) \text{ where } 1 \leq i \leq pr(\varphi), \psi = pred(\varphi, i)$$

This implies that

$$evaluate(replacePred(\varphi_{\bar{Q}}, entry(\bar{a})) = true \text{ iff } H \vDash \varphi_{\bar{Q}}(\bar{a})$$

The equivalence between the two cases of the interpretation and the rules 6 and 7 of the *FO*-semantics are obvious as well.

4 From tables to trees

Looking at the definition of evaluation tables it is very unlikely for two different hypergraphs to yield the same table. However we want to find a form of representation where sets(languages) of hypergraphs can be specified by one single value.

4.1 Evaluation trees

The idea of representing a table as a tree is nothing new. One can simply create a tree with one branching per dimension as equivalent to a table. This is what we plan to do with our evaluation tables.

Looking at the evaluation algorithm for tables we will notice that it now resembles searching a specific path in the corresponding tree. Assigning an additional variable then corresponds to picking a successor in the tree.

Definition 15 (Anonymous vertices)**.** *Simply writing a table as a tree will not solve our problem of finding a form representing that's suitable for representing languages of hypergraphs. To acquire this form we exploit the fact that any given FO can only distinguish a certain amount of vertices. If a sufficient number of vertices with a desired property exists, we don't need to know if there are any more of them. Therefore we try to find an upper bound of vertices that have to be considered.*

Furthermore we know that internal vertices of a terminal hypergraph are final. That means even if this hypergraph is embedded into another one there will never be additional hyperedges attached to this vertex. This knowledge is used to define anonymous vertices. An anonymous vertex is a vertex with no known identity. The only thing known is a set of properties and the fact that he does exist somewhere. As no hyperedge attached to this vertex can be added its exact position within the hypergraph does not matter anymore.

Definition 16 (The blank vertex)**.** *The blank vertex is a concept related to anonymous vertices. The blank vertex (denoted by $\square$) is also a placeholder for an unidentified vertex. However while anonymous vertices were known at some point the blank vertex represents a vertex that may occur in upcoming hyperedge replacements. As it was not identified yet there are no properties known either. It is just a vertex with currently no hyperedges.*

Definition 17 (Evaluation tree)**.** *Evaluation trees are meant to represent sets of terminal hypergraphs. The only thing these hypergraphs must have in common is the number of external vertices. Therefore we define evaluation trees depending on the formula and the number of external vertices as $ETree(\varphi, ext)$.*

The height of a tree is in analogy to tables the number of quantifiers $qr(\varphi)$ and the entries that are now stored in the leafs are of the form $\{true/false\}^{pr(\varphi)}$ too. Therefore we simply define evaluation trees of height 0 as such vectors. For a quantifier free formula ψ the set of trees is:

$$ETree(\psi, ext) = \{true, false\}^{pr(\psi)}$$

We increment the height of the tree by adding quantifiers to the formula. If we wanted to reproduce tables exactly we would have one successor for every possible vertex assignment. However we want to prevent redundancy in the entries. Therefore we anonymize vertices if possible. Every vertex besides the external ones can be anonymized. For external vertices the set of attached hyperedges is not final as new ones might be added by embedding the hypergraph in another graph. The blank vertex can obviously not be anonymized either as the set of hyperedges can not be determined before the identity of the vertex is known. Internal vertices can be anonymized though. This includes that we have to keep only one of them saved if they represent the exact same properties.

For $\varphi = Qx\ \psi$ we define elements $t \in ETree(\varphi, ext)$ as

$$t = (S, s_1, ..., s_{ext}, s_b)$$

The first entry is a set of anonymous successors. The corresponding original vertices do not need to be stored as their identity is not of interest for the evaluation. As explained the remaining vertices have to be treated separately. Therefore s_i is a tree belonging to the branching induced by the external vertex i. The tree s_b denotes the same for the blank vertex $\square$. The type of the successors is

$$S \subseteq ETree(\psi, ext)$$

and

$$s_1, ..., s_n, s_b \in ETree(\psi, ext)$$

4.2 Generating evaluation trees

To generate an evaluation tree to a given terminal hypergraph we have to look at possible assignments as well. The difference to evaluation trees is that assignments of anonymous vertices are not stored with the corresponding vertex id and might be merged.

Given a terminal hypergraph $H = (n, E, att, lab, ext)$ and an *FO*-formula φ we generate the evaluation tree $tree(H, \varphi) \in ETree(\varphi, ext)$ as follows:

$$tree(H, \varphi) = tree'(H, \varphi, \epsilon)$$

The last parameter is an vertex assignment needed to calculate the entries. The generation function is again defined recursive. If the formula φ is quantifier free ($qr(\varphi) = 0$) we are in the base case.

$$tree'(H, \varphi, \bar{a}) = evalPred(1, \bar{a}) \bullet ... \bullet evalPred(pr(\varphi), \bar{a})$$

where $evalPred$ is taken from the table generation.

If φ is not quantifier free ($\varphi = Qx\ \psi$) we have to branch the tree

$$tree'(H, \varphi, \bar{a}) = (S, s_1, ..., s_{ext}, s_b)$$

The successors for the external vertices and the blank one are straight forward

$$\begin{aligned} s_i &= tree'(H, \varphi, \bar{a} \bullet i) \quad \text{for } i \in \underline{ext} \\ s_b &= tree'(H, \varphi, \bar{a} \bullet \square) \end{aligned}$$

Accordingly the successors for anonymous vertices are created. However they are put in one set so that equivalent successors are implicitly merged.

$$S = \{tree'(H, \varphi, \bar{a} \bullet v) \mid v \in \underline{n} \setminus \underline{ext}\}$$

Example 8 (Evaluation tree generation). Figure 8 shows an example evaluation tree generated for the displayed hypergraph and formula. The procedure is the same like for tables. We take a look at possible assignments and add the according entries. One remarkable thing is that fact that vertex 2 is an internal vertex and therefore anonymized. This is indicated by the missing label on the tree branching.

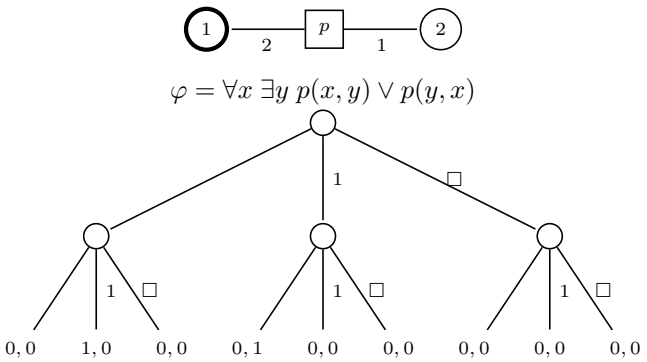


Figure 8: Example for tree generation

Figure 9 depicts the way *evalTree* works. Starting from the initial evaluation tree first the blank vertex successors are removed. Then the vectors in the leafs are evaluated. Then by iteratively adding quantifiers the tree's height is reduced in every step. As our first quantifier is an existential one the one *true* value in each branch suffices to yield a *true* value. Then the universal quantifier returns *true* as well (both successors are *true*).

4.3 Interpretation of evaluation trees

In Accordance with the evaluation tables we define a method to evaluate the evaluation tree. We can reuse the *replacePred* function defined in the section

"Interpretation of evaluation tables". As for the tables this function is necessary to condense entries into a single *true*/*false* value.

Let $t \in ETree(\varphi, ext)$ be an evaluation tree.

If φ is quantifier free ($qr(\varphi) = 0$) then

$$evalTree(t) = evaluate(replacePred(\varphi, t))$$

Otherwise φ has the form $Qx\,\psi$ with $Q \in \{\forall, \exists\}$ and we distinguish two cases.
Case 1: $Q = \forall$

$$evalTree(t) = \begin{cases} false & \text{if } evalTree(s) = false \text{ for an } s \in S \\ false & \text{if } evalTree(s_i) = false \text{ for an } i \in \underline{ext} \\ true & \text{otherwise} \end{cases}$$

Case 2: $Q = \exists$

$$evalTree(t) = \begin{cases} true & \text{if } evalTree(s) = true \text{ for an } s \in S \\ true & \text{if } evalTree(s_i) = true \text{ for an } i \in \underline{ext} \\ false & \text{otherwise} \end{cases}$$

Cases where variables are unassigned ($x = \square$) are not taken into account. The result of $evalTree(t)$ is a *true*/*false* constant and $evalTree(tree(H, \varphi)) = true$ iff $H \vDash \varphi$ holds.

Example 9 (Interpretation of a tree).

4.4 Correctness

We now want to show that for every hypergraph H, formula φ and assignment $\bar{a}$

$$H \vDash \varphi(\bar{a}) \text{ iff } evalTree(tree'(H, \varphi, \bar{a})) = true$$

This can be proven by induction over the quantifier rank of φ.

Base case: $qr(\varphi) = 0$

$$\begin{aligned} evalTree(tree'(H, \varphi, \bar{a})) &= evalTree(evalPred(1, \bar{a}) \bullet ... \bullet evalPred(pr(\varphi), \bar{a})) \\ &= evaluate(replacePred(\varphi, evalPred...)) \\ &= evaluate(\varphi(\bar{a})) \end{aligned}$$

Therefore the base case is correct.

Induction hypothesis: our assumption is correct for formulas with quantifier

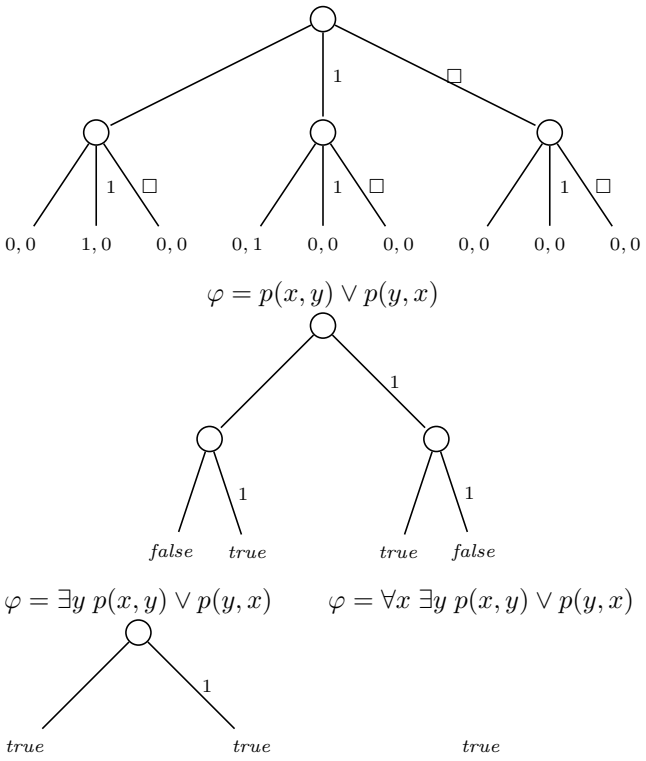


Figure 9: Example for tree evaluation

rank less than n. Let φ be a formula with $qr(\varphi) = n$ and $\varphi = \forall x\ \psi$

$$
\begin{aligned}
evalTree(tree'(H, \varphi, \bar{a})) &= evalTree((S, s_1, ..., s_{ext}, s_b)) \\
&= \begin{cases} \mathit{false} & \text{if } evalTree(s) = \mathit{false} \text{ for an } s \in S \\ \mathit{false} & \text{if } evalTree(s_i) = \mathit{false} \text{ for an } i \in \underline{ext} \\ \mathit{true} & \text{otherwise} \end{cases} \\
&= \begin{cases} \mathit{false} & \text{if } evalTree(tree'(H, \varphi, \bar{a} \bullet a)) \\ & = \mathit{false} \text{ for an } ext < a \leq n \\ \mathit{false} & \text{if } evalTree(tree'(H, \varphi, \bar{a} \bullet i)) \\ & = \mathit{false} \text{ for an } i \in \underline{ext} \\ \mathit{true} & \text{otherwise} \end{cases} \\
&= \begin{cases} \mathit{false} & \text{if } H \not\models \varphi(\bar{a} \bullet a) \text{ for an } ext < a \leq n \\ \mathit{false} & \text{if } H \not\models \varphi(\bar{a} \bullet i) = \mathit{false} \text{ for an } i \in \underline{ext} \\ \mathit{true} & \text{otherwise} \end{cases}
\end{aligned}
$$

This is equivalent to rule 6 of the *FO*-semantics. This equally works for $\exists$ and rule 7l. Therefore the assumption holds for formulas with quantifier rank n.

Hence we proved by induction that it holds for every formula.

from the definition of *tree*

$$H \vDash \varphi \text{ iff } evalTree(tree(H, \varphi)) = true$$

follows.

4.5 Finiteness of $ETree(\varphi, ext)$

One important result that is necessary to guarantee the termination of our upcoming algorithms it the fact that for a given *FO*-formula φ and number ext of external vertices the set of evaluation trees $ETree(\varphi ext)$ is finite. We will prove this fact by constructing the set.

For a quantifier free formula φ (height 0) $ETree(\varphi, ext)$ equals the set of possible vectors.

$$ETree(\varphi, ext) = \{true, false\}^{qr(\varphi)}$$

and

$$|ETree(\varphi, ext)| = 2^{qr(\varphi)}$$

When incrementing the height of the tree by one we have to try every possible combination of successor nodes. This means as anonymous successors we get an arbitrary subset for possible evaluation trees. Additionally we have to chose every possible assignment of evaluation trees for the special successors $s_1, ..., s_{ext}$ and s_b. The height of all these trees has of course to be 1 less. If $\varphi = Qx\,\psi$ where $Q \in \{\exists, \forall\}$ we get

$$ETree(\varphi, ext) = \left\{ (S, s_1, ..., s_{ext}, s_b) \mid \begin{array}{l} S \subseteq ETree(\psi, ext), \\ s_i \in ETree(\psi, ext) \text{ for } i \in \underline{n}, \\ s_b \in ETree(\psi, ext) \end{array} \right\}$$

The number of elements is

$$|ETree(\varphi, ext)| = |ETree(\psi, ext)|^{(ext+1)} \cdot 2^{|ETree(\psi, ext)|}$$

As long as ext and $qr(\varphi)$ are finite the set $ETree(\varphi, ext)$ is finite as well.

5 Evaluation for nonterminal hypergraphs

In this section we again examine a hypergraph $H = (n, E, att, lab, ext) \in HG_\Sigma$, a hyperedge replacement grammar $G \in HRG_\Sigma$ and a *FO* formula φ. Like before we want to generate an evaluation tree for H in order to check whether this hypergraph satisfies φ. However this time the hypergraph H contains nonterminal hyperedges. To generate an evaluation tree representing this hypergraph we have to look at possible derivations.

5.1 Evaluation of symbols

So far we only created evaluation trees for hypergraphs. Our goal however is to be able to provide evaluation trees for nonterminal symbols as well. The desired mapping $tree : X \in N, \varphi \in FO \rightarrow ETree(\varphi, rk(X))$ has the characteristic that for every derivation

$$X \overset{G}{\Longrightarrow}^* K$$

of a terminal hypergraph K the evaluation tree of K is given by

$$tree(K, \varphi) = tree(X, \varphi)$$

For now we consider this function to be given. In subsequent section we will discuss how to modify hyperedge replacement grammars in order to define them.

First we want to study how hyperedge replacement influences the evaluation for the embedding hypergraph. When replacing hyperedges new vertices and hyperedges may be added to the embedding hypergraph. This definitely influences the evaluation. If for every nonterminal hyperedge information about possible derivations is given as evaluation tree we are able to treat the hypergraph as terminal hypergraph and generate one unambiguous evaluation tree.

Definition 18 (Evaluation tree for nonterminal hypergraphs)**.** *We want to provide a similar definition of evaluation trees for nonterminal hypergraphs. It's basically the same like for terminal hypergraphs. The only difference is that internal vertices are not anonymous. This is because they can be attached to new hyperedges by replacing the existing nonterminal ones. Therefore the set of trees is dependent on the total number of vertices.*

As the definition of $NETree(\varphi, n)$ is nearly equivalent to $ETree(\varphi, ext)$ we will not discuss it in detail. For a quantifier free formula ψ the set of trees is:

$$ETree(\psi, n) = \{true, false\}^{pr(\psi)}$$

For $\varphi = Qx\ \psi$ we define elements $t \in NETree(\varphi, n)$ as

$$t = (S, s_1, ..., s_n, s_b)$$

with

$$\begin{aligned} S &\subseteq ETree(\psi, ext) \\ s_1, ..., s_n, s_b &\in ETree(\psi, ext) \end{aligned}$$

5.2 Generating evaluation trees for nonterminal hypergraphs

This procedure also equals it pendant for terminal hypergraphs besides the different interpretation of anonymous vertices.

Given a terminal hypergraph $H = (n_H, E, att, lab, ext)$ and an FO-formula φ we generate the evaluation tree $nTree(H, \varphi) \in NETree(\varphi, n_H)$ as follows:

$$nTree(H, \varphi) = nTree'(H, n_H, \varphi, \epsilon)$$

The second parameter defines how many vertices need to stay concrete. Thre rest is anonymized. If the formula φ is quantifier free ($qr(\varphi) = 0$) we are in the base case.

$$nTree'(H, n, \varphi, \bar{v}) = evalPred(1, \bar{a}) \bullet ... \bullet evalPred(pr(\varphi), \bar{a})$$

where $evalPred$ is taken from the table generation.

If φ is not quantifier free ($\varphi = Qx\ \psi$) we have to branch the tree

$$tree'(H, \varphi, \bar{v} = (S, s_1, ..., s_n, s_b)$$

The successors for the external vertices and the blank one are straight forward

$$\begin{aligned} s_i &= nTree'(H, n, \varphi, \bar{v} \bullet i) \quad \text{for } i \in \underline{n} \\ s_b &= nTree'(H, n, \varphi, \bar{v} \bullet \square) \end{aligned}$$

And for the vertices with id $> n$ we create an anonymous successor

$$S = \{nTree'(H, n, \varphi, \bar{a} \bullet i) \mid n < i \leq n_H\}$$

5.3 Impact of hyperedge replacement

The tree generated above would be correct if every hyperedge in H was removed without replacement. We are now interested how this tree has to be changed if we replace one single nonterminal hyperedge.

Let $H = (n, E, att, lab, ext)$ be a hypergraph, φ be an FO-formula and $t_H = nTree(H, \varphi)$ be the nonterminal evaluation tree. For a nonterminal hyperedge $e \in E$ with $lab(e) = X$. We consider an evaluation tree $t_X = tree(X, \varphi)$ to be given with

$$tree(K, \varphi) = t_X$$

for every derivation

$$X \overset{G}{\Longrightarrow}^* K$$

We now want to create an evaluation table t' with

$$nTree(H[e/K], \varphi) = t'$$

for each of these derivations. This is done by the function $merge$ which take both trees and an attachment to calculate t'

$$t' = merge(t_H, t_X, att(e))$$

To merge these trees we have to consider every combination of paths(assignments). Merging leafs is simply done by or-combination (denoted by $+$) of the boolean vectors. For $t_1 \in NETree(\psi, n), t_2 \in ETree(\psi, ext)$ with $qr(\psi) = 0$

$$merge(t_1, t_2, \bar{a}) = t_1 + t_2$$

In any other case we have $t_1 \in NETree(\psi, n), t_2 \in ETree(\psi, ext)$ with $\psi = Qx\ \omega$. Then

$$t_1 = (S_1, s_{(1,1)}, ..., s_{(1,n)}, s_{(1,b)})$$
$$t_2 = (S_2, s_{(2,1)}, ..., s_{(2,ext)}, s_{(2,b)})$$

Now we have to think about how these successors can be combined into

$$merge(t_1, t_2, \bar{a}) = (S, s_1, ..., s_n, s_b)$$

We start with the anonymous vertices. Obviously every anonymous vertex from t_1 is unknown to t_2 and vice versa. Choosing an anonymous successor in one of the trees forces us to take the blank successor in the other one.

$$S = S_{A1} \cup S_{A2} \text{ where } \begin{array}{ll} S_{A1} & = \{merge(s, s_{(2,b)}, \bar{a}) \mid s \in S_1\} \\ S_{A2} & = \{merge(s_{(1,b)}, s, \bar{a}) \mid s \in S_2\} \end{array}$$

The blank successor is also obviously

$$s_b = merge(s_{1,b}, s_{2,b}, \bar{a})$$

Finally we have to deal with the non anonymous successors we have to use the assignment in order to map vertices in t_1 on external vertices of t_2. The result is

$$s_v = merge(s_{1,v}, s, \bar{a}) \text{ where } s = \begin{cases} i & \text{if } \bar{a}[i] = v \\ \square & \text{otherwise} \end{cases}$$

Example 10 (Merging trees). Figure 10 sows the merging of two evaluation tables. On the left hand side we have the nonterminal evaluation tree for the hypergraph depicted above. The tree on the right hand side is for the nonterminal symbol X. The evaluation tree (c) shows the merging result representing every hypergraph derivable from the initial hypergraph.

5.4 Terminal evaluation of nonterminal hypergraphs

Given a nonterminal hypergraph $H = (n, E, att, lab, ext)$ with nonterminal hyperedges $e_1, ..., e_k$ we are able to generate a terminal evaluation tree iff every label $X_i = lab(e_i)$ has an associated evaluation tree $t_i = tree(X_i, \varphi)$

This terminal evaluation by merging every hyperedge

$$t = merge(...merge(nTree(H, \varphi), t_1, att(e_1))..., t_k, att(e_k))$$

and then evaluating

$$t_H = nt2t(t)$$

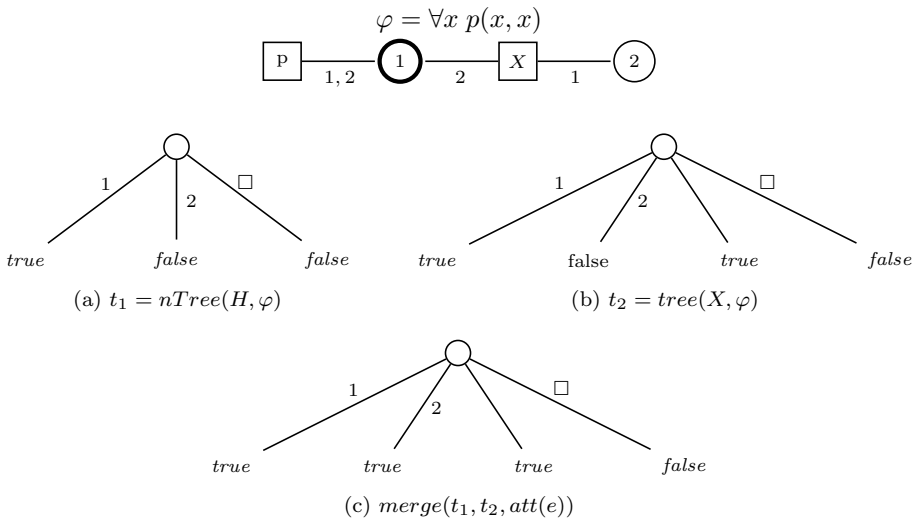


Figure 10: Example for merging of trees

where

$$nt2t(t) = \begin{cases} t & \text{if } t \in nETree(\varphi, n) \text{ and } qr(\varphi) = 0 \\ (S', s_1, ..., s_{ext}, s_b), & \text{if } t = (S, s_1, ..., s_n, s_b) \\ S' = S \cup \{s_i \mid ext < i \leq n\} & \end{cases}$$

We call this procedure $terminalTree(t)$

5.5 Correctness

Let H be a nonterminal hypergraph, φ be a formula and $e \in E_H$ be a nonterminal hyperedge. We define mapping functions

$$map_H(k) = \begin{cases} k & \text{if } k \leq n_H \\ \square & \text{otherwise} \end{cases}$$

$$map_H(k) = \begin{cases} k + ext_K & \text{if } k > n_H \\ i & \text{if } att(e)[i] = k \\ \square & \text{otherwise} \end{cases}$$

We want to prove that for any terminal hypergraph K with $ext_K = rk(e)$ and $\bar{v} \in V^*$ where V is the set of vertices in $H[e/K]$.

$$\begin{aligned} &merge(nTree'(H, n_H, \varphi, map_H(\bar{v})), tree'(K, \varphi, map_K(\bar{v})), att(e)) \\ &= nTree'(H[e/K], n_H, \varphi, \bar{v}) \end{aligned}$$

We again start an induction over the quantifier rank of φ.

Base case $qr(\varphi) = 0$:

for the left hand side we get

$$evalPred_H(1, map_H(\bar{a})) \bullet ... + evalPred_K(1, map_K(\bar{a})) \bullet ...$$

and for the right hand side:

$$evalPred_{H[e/K]}(1, \bar{a}) \bullet ...$$

The left hand sides evalues to *true* iff an according hyperedge exists in H or in K and the right hand side evaluates to *true* if an according hyperedge exists in $H[e/K]$. Therefore both expressions are equal.

The induction hypothesis is that our assumption holds for every formula with quantifier rank less than n.

Let φ be an formula with $qr(\varphi) = n$ and $\varphi = Qx$. Then both sides have the form

$$(S, s_1, ..., s_n, s_b)$$

Let's compare the sets S. After some valid simplifications the left hand side evaluates to

$$S = \left\{ \begin{array}{l} merge((nTree'(H, n_H, \varphi, map_H(\bar{v})), tree'(K, \varphi, map_K(\bar{v})), att(e)) \\ \mid n_H < v \leq n_H + n_K - ext_K \end{array} \right\}$$

When using the induction hypothesis the yields the evaluation of the right hand side.

Now we evaluate the left hand side value for s_i for each $i \in \underline{n_H} \cup \{\square\}$

$$s_i = merge(nTree'(H, n_H, \psi, map_H(\bar{v} \bullet i)), tree'(K, \psi, map_K(\bar{v} \bullet i)), att(e))$$

From the induction hypothesis we know that this equals

$$nTree'(H[e/K], n_H, \psi, \bar{v} \bullet i)$$

which is exactly the value of s_i on the right hand side.

By induction the assumption holds for every formula.

The above result directly yields the important conclusion

$$merge(nTree(H, \varphi, n_H i), tree(K, \varphi), att(e)) = nTree(H[e/K], \varphi, n)$$

Applying this knowledge on the *terminalTree* function implies that for every set of replacing hypergraphs $K_1, ..., K_k$ with $tree(K_i, \varphi) = tree(X_i, \varphi)$

$$terminalTree(H) = tree(H[e_1/K_1]...[e_k/K_k], \varphi)$$

This exactly the property for *terminalTree* that we need.

6 Modifying hyperedge replacement grammars

In this section we introduce a few operations on hyperedge replacement grammars that are used in the following algorithms. The most important fact about these operations is that while changing the grammar itself the language of the resulting grammar remains the same.

6.1 Removing non-productive rules

It's possible to create hyperedge replacement grammars that contain non-productive rules. A non-productive rule has a right hand side from which no terminal hypergraph can be derived. This happens if the derivation runs into a cyclic sequence of rules without the possibility to chose a terminal rule at any point. Obviously these rules don't add anything to the language as no hypergraph can be generated using them. Therefore the hyperedge replacement grammar obtained by removing these rules has the same language. This removal can be done by using a simple algorithm:

1. Mark terminal rules (terminal right hand side) as productive.
2. Mark nonterminal symbols with a marked rule as productive.
3. Mark rules that contain only nonterminal hyperedges with a marked label.
4. if something new was marked go back to 2.
5. remove non-productive nonterminal symbols and rules.

Let the resulting hyperedge replacement grammar be G'.

To fact that this modification does not alter the language can be shown by induction over the derivation length. We want to show that for any terminal $H \in HG_\Sigma$, $G \in HRG_\Sigma$, $X \in N$ and $n \in \mathbb{N}$

$$X \overset{G}{\Longrightarrow}^n H \text{ implies } X \overset{G'}{\Longrightarrow}^n H$$

Starting with $n = 1$ we know that this derivation consists of only one terminal rule. As terminal rules are always marked it is still in G'. Therefore our assumption is correct for $n = 1$.

The induction hypothesis is that our assumption holds for every $m < n$. Given a derivation $X \overset{G}{\Longrightarrow}^n H$ we know it has the form

$$X \overset{x \to K}{\Longrightarrow} K \overset{G}{\Longrightarrow}^{n-1} H$$

From the induction we then conclude

$$X \overset{x \to K}{\Longrightarrow} K \overset{G'}{\Longrightarrow}^{n-1} H$$

This implies that every nonterminal symbol in K has been marked. Therefore $x \to K$ was marked as well. Hence

$$X \overset{G'}{\Longrightarrow}^{n} H$$

which proves our assumption for n. By induction this proves that $\mathcal{L}(G) = \mathcal{L}(G')$.

6.2 Splitting of nonterminals

Another way to modify hyperedge replacement grammars is to split nonterminal symbols. When we split X a nonterminal symbol we create a new infant nonterminal X' which is equally derivable. This means if we are able to derive a hypergraph with an X-labeled hyperedge e then we have to be able to derive the same hypergraph with $lab(e) = X'$.

Given a hyperedge replacement grammar $G = (N, T, P, S)$ we define the splitting as

$$split(G, X) = G' = (N \cup \{X'\}, T, P', S)$$

To modify the production rules we simply have to put X' in every possible position where X occurs. Let $p = Y \to H$ where $X \in P$ be a production rule where X occurs $n = |\{e \in E_H \mid lab_H(e) = X\}|$ times on the right hand side. We consider the concerned hyperedges to be ordered $(e_1, ..., e_n)$. We then define

$$H'(\bar{x}) = (n_H, E_H, att_H, lab', ext_H) \text{ where } lab'(e) = \begin{cases} \bar{x}[i] & \text{if } e = e_i \\ lab_H(e) & \text{otherwise} \end{cases}$$

and then

$$newRules(p) = \left\{Y \to H'(\bar{x}) \mid \bar{x} \in \{X, X'\}^n\right\}$$

The set of new production rules can finally be defined as

$$P' = \{newRules(p) \mid p \in P\}$$

This modification has obviously no impact on the language as the added symbol is not productive.

Now we allow to assign rules for X to X'. This means if we have a grammar $G = (N, T, P, S)$ we create a new grammar $G' = (N, T, P', S)$. Where for one production rule is reassigned. Let $X \to H \in P$ be a production rule and X' an infant nonterminal of X. Then

$$P' = P \setminus \{X \to H\} \cup \{X' \to H\}$$

This dos not change the language as well as for every derivation

$$K \overset{G}{\Longrightarrow}^{*} L \overset{X \to H}{\Longrightarrow} M \overset{G}{\Longrightarrow}^{*} N$$

a derivation

$$K \overset{G'}{\Longrightarrow}^{*} L' \overset{X' \to H}{\Longrightarrow} M \overset{G'}{\Longrightarrow}^{*} N$$

can be calculated. This follows from the definition of splitting.

7 Algorithm 1: Straight forward

In this section we will introduce a first algorithm to calculate for a given grammar $G = (N, T, P, S) \in HRG_\Sigma$ whether the language $\mathcal{L}(G)$ has a certain property defined by φ. Previously we considered the evaluation for the nonterminal symbols to be given. Now we have to provide a strategy to calculate these evaluation trees.

This brings up one main problem. For nonterminal symbols with more than one rule it's very likely to calculate different evaluation trees for different rules. If this happens we're unable to give an unambiguous result for this nonterminal. This however is required in order to apply the evaluation strategies we introduced before. To deal with this ambiguity we have to somehow create more different nonterminal symbols until we get to a point where we can give one single evaluation tree for each of them. At this point the finiteness result for $\mathcal{C}$ becomes very important. As we have to create at most one nonterminal for every possible evaluation (with respect to equivalence of trees), it provides us with an upper bounds for the number of symbols we have to create. This directly leads us to the idea for a first approach.

7.1 Initial grammar

One first and simple possibility is to generate a new grammar containing every potential nonterminal symbol in advance and then successively remove every one of them that is not needed.

For the new hyperedge replacement grammar $G' = (N', T, P', S')$ we generate a nonterminal for every possible evaluation tree. For a nonterminal symbol $X \in N$ we create one nonterminal symbol $(X, t) \in N'$ for every evaluation tree $t \in ETree(\varphi, rk(X))$.

To make sure every eventuality is covered every created nonterminal symbol inherits every rule from the nonterminal it was derived from. However we have to consider the new possible combinations of nonterminals on the right hand side. For every nonterminal $(X, t) \in N'$ and every rule $X \to K \in P$ we get new rules according to the nonterminal hyperedges occurring in K. Let $e_1, ..., e_k$ be the nonterminal hyperedges in K. For every combination of new labels $l_1 \in \{Y \in N' \mid Y = (lab_K(e_1), t)\}$, ..., $l_k \in \{Y \in N' \mid Y = (lab_K(e_k), t)\}$ we add a rule $(X, t) \to K'$ with $K' = (n_K, E_K, att_K, lab', ext_K)$ and

$$lab'(e) = \begin{cases} l_i & \text{if } e = e_i \\ lab(e) & \text{otherwise} \end{cases}$$

Finally we create rules from the new starting symbol S' to every symbol (S, t) derived from the former starting symbol

$$S' \to K = (0, \{e\}, \emptyset, \{e \to (S, t)\}, 0)$$

This new hyperedge replacement grammar obviously has the same language as the initial one.

7.2 Reasonable symbols and rules

Our new hyperedge replacement grammar G' contains a very large amount of rules and nonterminals. Our goal is to only have nonterminals (X, t) where the associated evaluation table is reasonable. Reasonability in this context implies that the evaluation for every derivable hypergraph results in t. More formal: $(X, t) \in N'$ is reasonable iff

$$tree(H, \varphi) = t \text{ for every} (X, t) \overset{G'}{\Longrightarrow}{}^* H$$

Accordingly we define reasonability for hyperedge replacement rules: $(X, t) \to K \in P'$ is reasonable iff

$$tree(H, \varphi) = t \text{ for every } K \overset{G'}{\Longrightarrow}{}^* H$$

This directly implies an equivalence between reasonability of a nonterminal symbol and the reasonability of all its rules.

The vast majority of the generated nonterminals and rules is not reasonable. In fact in most cases the described generation of G' wont contain a single reasonable nonterminal symbol.

7.3 Minimizing the grammar

Using the above definition of reasonability we now try to create a second hyperedge replacement grammar G'' which contains only reasonable nonterminal symbols while the language still stays intact.

To achieve this we use an algorithm that's similar to the one we introduced for removing non-productive rules and symbols. We just replace the productivity by reasonability.

1. Mark the starting symbol S' as reasonable
2. Mark terminal rules $(X, t) \to H$ as reasonable if $tree(H, \varphi) = t$.
3. Mark nonterminal symbols with a marked rule as reasonable.
4. Mark rules $(X, t) \to H$ if H only contains nonterminal hyperedges with marked label and $tree(H, \varphi) = t$.
5. if something new was marked go back to 3.
6. Mark rules of S' for marked $(S, t) \in N'$)
7. remove unmarked nonterminal symbols and rules.

Step 3 might need some explanation. The evaluation of H is done by applying $terminalTree(nTree(H, \varphi))$. As precalculated result $tree(lab(e))$ for a nonterminal hyperedge e with label $lab(e) = (X, t)$ the evaluation tree t is used.

Application of the algorithm now yields a hyperedge replacement grammar that contains only reasonable nonterminal symbols.

7.4 Correctness

First we want to prove that removing hyperedges like this does not change the language of the resulting grammar. Therfore we have to prove that $\mathcal{L}(G'') \subseteq \mathcal{L}(G)$ and $\mathcal{L}(G'') \supseteq \mathcal{L}(G)$.

Step 1 "$\subseteq$":
As G'' is obtained by removing production rules from G' obviously $\mathcal{L}(G'') \subseteq \mathcal{L}(G')$ holds. Therefore showing even $\mathcal{L}(G')$ is a subset of $\mathcal{L}(G'')$ implies the desired relation. To prove this we take a look at hypergraphs derivable from S'. For every terminal hypergraph K with $S' \overset{G'}{\Longrightarrow}{}^* K$ we take a look at the associated sequence of hyperedge replacement rules. If we find corresponding rules in P a derivation $\overset{G}{\Longrightarrow}{}^* K$ exists as well. This can be done applying rule the rule $X \to H'$ for G whenever a rule $(X,t) \to H$ is used in G' where H' is obtained from H by modifying the label function so that $lab'(e) = X$ if $lab(e) = (X,t)$ and $lab'(e) = lab(e)$ otherwise. The only exception is the starting symbol S'. Because it is just used to "pick" from different starting symbols this step can be neglected in G. As this is a valid derivation we thereby proved that $\mathcal{L}(G'') \subseteq \mathcal{L}(G)$.

Step 2 "$\supseteq$":
We claim that for every nonterminal symbol $X \in N_G$ of the initial grammar and every terminal hypergraph K derivable from it ($X \overset{G}{\Longrightarrow}{}^* K$) this hypergraph can be derived in G'' from the nonterminal symbol (X,t) where $t = tree(K,\varphi)$. This can be proved by an induction over the derivation length.

The base case is $n = 1$:

If the derivation has length 1 we're dealing with a terminal rule $X \to H$. From step 2 of the algorithm we can see that terminal $(X,t) \to H$ rules are marked and therefore remain in the grammar G'' iff $tree(H,\varphi) = t$. This implies that our prediction works for the base case.

For the induction step we assume that $(X, tree(K,\varphi)) \overset{G''}{\Longrightarrow}{}^m K$ for every terminal hypergraph K, nonterminal symbol $X \in N_G$ and $m < n$ where $X \overset{G}{\Longrightarrow}{}^m K$.

Now given a derivation $X \overset{G}{\Longrightarrow}{}^n K$ we take a look at the first rule $X \to H$. We know that $X \overset{X\to H}{\Longrightarrow} H \overset{G}{\Longrightarrow}{}^{n-1} K$. We can conclude that $K = H[e_1/K_1]...[e_k/K_k]$ and $X_i \overset{G}{\Longrightarrow}{}^{m_i} K_i$ where e_i is a nonterminal hyperedges of H, $X_i = lab(e_i)$ is its lable and $m_i < n$ for all $i \in \underline{k}$. Using our induction hypothesis this implies that $(X_i, tree(K_i,\varphi)) \overset{G''}{\Longrightarrow}{}^{m_i} K_i$ for $i \in \underline{k}$. Therefore all the symbols $(X_i, tree(K_i,\varphi))$ have at least one reasonable rule and are therefore reasonable themselves. Knowing this steps 4 and 3 of the algorithm imply that $(X, tree(K,\varphi)) \overset{G''}{\Longrightarrow}{}^n K$.

It has been proved by mathematical induction that $(X, tree(K,\varphi)) \overset{G''}{\Longrightarrow}{}^* K$ for every $X \overset{G}{\Longrightarrow}{}^* K$

As we have rules from S' to every remaining (S,t) in G'' this implies that

$S' \overset{G''}{\Longrightarrow}{}^* K$ or every $S \overset{G}{\Longrightarrow}{}^* K$ which proves the desired $\supset$-relation.

The second step also provides us with another important proof. It also proves that for every starting symbol $(S,t) \in N''$ t defines the evaluation table of every derivable hypergraph:

$$tree(K,\varphi) = t \text{ for every } (S,t) \overset{G''}{\Longrightarrow}{}^* K$$

7.5 Analysis of G''

The analysis of G'' gives us the desired information about the language defined by G. We only have to look at the set of remaining starting symbols $\mathcal{S} = \{(S,t) \mid (S,t) \in N''\}$. The whole language is composed of the hypergraphs derived from these symbols:

$$\mathcal{L}(G) = \bigcup_{X \in \mathcal{S}} \left\{ K \mid X \overset{G''}{\Longrightarrow}{}^* K \right\}$$

We furthermore know that for every hypergraph K with $(S,t) \overset{G''}{\Longrightarrow}{}^* 7K$ the evaluation $tree(K,\varphi)$ equals t. If we now evaluate these tables we get information about every hypergraph included in the language. For the language $\mathcal{L}(G)$ we conclude that...

... $G \vDash \varphi$ iff $evalTree(t) = true$ for every $(S,t) \in \mathcal{S}$.

... $G \nvDash \varphi$ iff $evalTree(t) = false$ for every $(S,t) \in \mathcal{S}$.

... $G \approx \varphi$ otherwise.

7.6 Complexity

Now we will have a short look at the complexity of this algorithm. The complexity of creating G' equals the count of its nonterminal symbols and production rules. The same holds for the second part of the algorithm as we remove at least one production rule during each iteration. The number of nonterminal symbols and production rules in turn depends on the initial hyperedge replacement grammar and on the formula.

However we are not looking for an exact value. Therefore we calculate a lower bound as number of new nonterminals for one nonterminal. We only approximate the number of nonterminals by approximating the number of possible evaluation tables $|ETree(\varphi, ext)|$. This number can be directly derived from the proof of the finiteness. We get

$$|ETree(\varphi, n)| = countT(qr(\varphi), n)$$

where

$$\begin{aligned} countT(0,n) &= 2^{pr(\varphi)} \\ countT(q,n) &= countT(q-1)^{n+1} \cdot 2^{countT(q-1)} \end{aligned}$$

To obtain a lower bound for this sequence we again use a strong approximation.

$$\begin{aligned} countT(0,n) &= 2^{pr(\varphi)} \\ countT(q,n) &\leq 2^{countT(q-1)} \end{aligned}$$

This leads to an approximation of the total runtime complexity as

$$\underbrace{2^{2^{\cdot^{\cdot^{2^{pr(\varphi)}}}}}}_{qr(\varphi)}$$

Even for this very strong underestimation we get a complexity that grows horribly with the quantifier rank of φ.

8 Algorithm 2: More sophisticated

We now take a look at our second approach. Previously we generated every possible nonterminal and then deleted the ones that were not reasonable. As this applies to the majority of symbols the idea is to generate only the needed ones. To achieve this we try to create an example derivation for every possible evaluation. The central idea is to split a nonterminal if different rules yield different evaluations.

8.1 The algorithm

Given a hyperedge replacement grammar G and a formula φ we want to generate a hyperedge replacement grammarG' with $\mathcal{L}(G) = \mathcal{L}(G)$ and an associated function $tree : X \in N' \to ETree(\varphi, rk(X))$. W.l.o.g. we consider G to be free of non-productive rules and symbols. Then successively look at every production rule. If we have precalculated evaluation tables for every nonterminal hyperedge on the right hand side we're generate the evaluation table for this hypergraph. Depending on the result we then split the nonterminal.

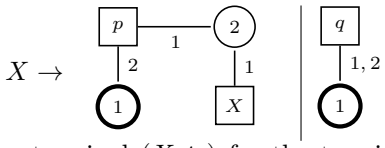
We define sets of splitting operations

1. For each terminal rule $X \to H$ split X creating an infant symbol $(X, tree(H))$ and assign the rule to $(X, tree(H))$.
2. If existent pick rule $X \to H$ where X is no infant symbol and for every nonterminal hyperedge $e \in E_H$ the label $lab(e)$ is an infant symbol.
3. Evaluate $terminalTree(nTree(H, \varphi))$ where the precalculated evaluation tree for a symbol (X, t) is $tree((X, t)) = t$.
4. Split X creating the infant symbol $(X, tree(H))$.
5. Assign the rule to $(X, tree(H))$.
6. Continue at 2. until every production rule was treated.

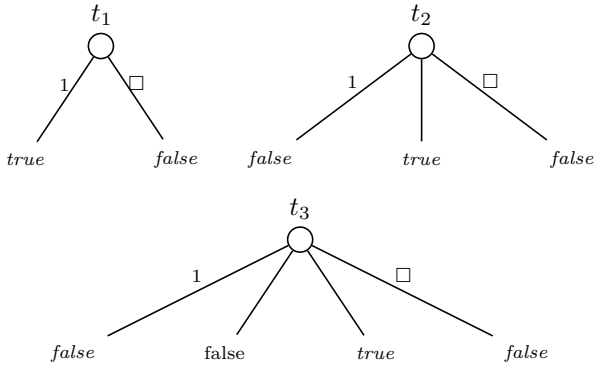
7. Remove every initial nonterminal symbol and rules where it occurs.

8. Add a new starting symbol S' which derives every symbol (S, t).

Example 11 (Example run of the algorithm). We will now have a look at a small example run of the algorithm. The formula is given as $\varphi = \exists x\, p(x, x)$. And the *HRG* is



The algorithm then creates a nonterminal (X, t_1) for the terminal rule. Then the nonterminal rule can be used to create the nonterminal (X, t_2) for the other rule. Inserting this in the right hand side finally yields (X, t_3).



8.2 Correctness

We first take a look at the hyperedge replacement grammar G' after step 6. We want to show that every initial nonterminal symbol is no longer productive. We prove it by an induction over the derivation length. For every nonterminal symbol $X \in N_G$ no terminal rule exists. This follows directly from step 1. Hence for length $n = 1$ no terminal hypergraph can be derived. Taking the step from $n - 1$ to n we look at an arbitrary derivation from X

$$X \overset{G'}{\Longrightarrow}^{n} H$$

This can be written as

$$X \overset{X \to K}{\Longrightarrow} K \overset{G'}{\Longrightarrow}^{n-1} H$$

Looking at step 2 - 5 we see that K has at least one hyperedge labeled with an non infant symbol Y (otherwise it would've been assigned to an infant). The induction hypothesis implies that no terminal hypergraph can be derived from Y in $n - 1$ or less steps. Therefore H is not terminal as well.

By induction we proved that each of the initial symbols is non-productive and

can be removed.

Let the grammar obtained at the end of the algorithm be G''. The language of G are still G'' the same (we only used splitting and removal of non-productive symbols which preserves the language). By induction over the derivation length we only have to show that for every derivable terminal hypergraph H

$$(X,t) \overset{G''}{\Longrightarrow}{}^{*} H \Rightarrow tree(H) = t$$

holds. For derivations of length $n = 1$ this again follows from step 1. For $n > 1$ we look at derivations

$$(X,t) \overset{(X,t)\to K}{\Longrightarrow} K \overset{G''}{\Longrightarrow}{}^{n-1} H$$

The equality

$$t = tree(H)$$

follows directly from the induction hypothesis and the steps 2-5. This is because the derivation equals a replacement of nonterminal hyperedges by terminal hypergraphs that are derivable from its label. As this derivation is definitely of length less than n we know that the tree $tree(X,t)$ for this label is correct (induction hypothesis) and this tree is used in step 3 the overall result is correct too.

This two proofs show that this algorithm has exactly the same result as the one presented before. Therefore the analysis of G'' works accordingly.

8.3 Complexity

Although the expected runtime in most practicable scenarios is much lower there might be cases where the worst case runtime of the first algorithm is need for this algorithm as well.

8.4 Further optimizations

For possible implementations of this algorithm some further optimizations are imaginable.

1. Detect values $\bar{v} \in \{true, false\}$ for which the evaluation is determined and not further *true* entry can alter the result. These values do not have to be distinguished. This can be done by using binary decision diagrams.

2. Evaluate leafs as soon as possible (e.g. if the path only consists of anonymous successors the values are final). Then quantifiers might be evaluated earlier. This leads to trees where the height may differ between different branches. Preevaluated nodes then contain a *true*/*false* value instead of successors. This increases the chance of equality between trees and leads to fewer nonterminal symbols.

9 Extending to *MSO*

MSO or Monadic second order logic is an extension of *FO* where quantification over sets of vertices is possible.

Definition 19 (*MSO* over hypergraphs)**.** *To define the syntax of MSO we extend FO by a few constructs. As we have to introduce set variables we define MSO as* $MSO(T, X, Y)$ *where* Y *is the set of set variables and is considered to be ordered as well.*

The atomic formulas of FO remain.

$$true, false \in MSO(T, X, Y) \text{ for any } T, X, Y$$

$$p(\bar{x}) \in MSO(T, X, Y) \text{ for any } Y \text{ if } p \in T, \bar{x} \in X^* \text{ and } rk(p) = |\bar{x}|$$

Now one new atomic formula is added

$$S(x) \in MSO(T, X, Y) \text{ for any } T \text{ if } S \in Y, x \in X$$

This atomic formula states that a certain vertex is in the set specified by S*.*

Now we will discuss the logical operations (negation, conjunction and disjunction)

$$\begin{aligned}
\neg\varphi &\in MSO(T, X, Y) \quad \text{iff } \varphi \in MSO(T, X, Y) \\
\varphi \wedge \psi &\in MSO(T, X, Y) \quad \text{iff } \varphi \in FO(T, X', Y'), \psi \in FO(T, X'', Y'') \\
&\qquad \text{and } X \supseteq X' \cup X'', Y \supseteq Y' \cup Y' \\
\varphi \vee \psi &\in MSO(T, X, Y) \quad \text{iff } \varphi \in FO(T, X', Y'), \psi \in FO(T, X'', Y'') \\
&\qquad \text{and } X \supseteq X' \cup X'', Y \supseteq Y' \cup Y'
\end{aligned}$$

Finally we include the quantifiers. Here the most significant change appears.

$$\begin{aligned}
\forall x\, \varphi &\in MSO(T, X, Y) \quad \text{iff } \varphi \in FO(T, X', Y) \text{ and } X \supseteq X' \setminus \{x\} \\
\exists x\, \varphi &\in MSO(T, X, Y) \quad \text{iff } \varphi \in FO(T, X', Y) \text{ and } X \supseteq X' \setminus \{x\} \\
\forall S\, \varphi &\in MSO(T, X, Y) \quad \text{iff } \varphi \in FO(T, X, Y') \text{ and } Y \supseteq Y' \setminus \{S\} \\
\exists S\, \varphi &\in MSO(T, X, Y) \quad \text{iff } \varphi \in FO(T, X, Y') \text{ and } Y \supseteq Y' \setminus \{S\}
\end{aligned}$$

quantification over set variables is identified by writing the variable in capital letters.

Definition 20 (Semantics of *FO*-formulas)**.** *Given a terminal hypergraph* $H \in HG_\Sigma$ *and a formula* $\varphi \in MSO(T, \emptyset, \emptyset)$ *we define a set of rules to determine whether* H *satisfies* φ *(*$H \vDash \varphi$*):*

1. $H \vDash \psi(\bar{a}, \bar{A})$ *if* $\psi = true$*.*
2. $H \vDash \psi(\bar{a}, \bar{A})$ *if* $\psi = p(\bar{x})$ *and a hyperedge* $e \in E_H$ *exists with* $att(e) = \bar{x} \times \bar{a}$*.*
3. $H \vDash \psi(\bar{a}, \bar{A})$ *if* $\psi = S(x)$ *and* $\bar{a}[x] \in \bar{A}[S]$*.*

4. $H \vDash \psi(\bar{a}, \bar{A})$ *if* $\psi = \neg\omega$ *and* $H \nvDash \omega(\bar{a}, \bar{A})$.
5. $H \vDash \psi(\bar{a}, \bar{A})$ *if* $\psi = \omega \wedge \gamma$ *with* $H \vDash \omega(\bar{a}, \bar{A})$ *and* $H \vDash \gamma(\bar{a}, \bar{A})$.
6. $H \vDash \psi(\bar{a}, \bar{A})$ *if* $\psi = \omega \vee \gamma$ *with* $H \vDash \omega(\bar{a}, \bar{A})$ *or* $H \vDash \gamma(\bar{a}, \bar{A})$.
7. $H \vDash \psi(\bar{a}, \bar{A})$ *if* $\psi = \forall x\, \omega$ *and* $H \vDash \omega(\bar{a} \bullet v, \bar{A})$ *for every* $v \in \underline{n_H}$.
8. $H \vDash \psi(\bar{a}, \bar{A})$ *if* $\psi = \exists x\, \omega$ *and* $H \vDash \omega(\bar{a} \bullet v, \bar{A})$ *for at least one* $v \in \underline{n_H}$.
9. $H \vDash \psi(\bar{a}, \bar{A})$ *if* $\psi = \forall S\, \omega$ *and* $H \vDash \omega(\bar{a}, \bar{A} \bullet M)$ *for every* $M \subseteq \underline{n_H}$.
10. $H \vDash \psi(\bar{a}, \bar{A})$ *if* $\psi = \exists S\, \omega$ *and* $H \vDash \omega(\bar{a}, \bar{A} \bullet M)$ *for at least one* $M \subseteq \underline{n_H}$.

9.1 Evaluation trees for *MSO*

Evaluation trees can be prepared to work with *MSO*. To do so we have to add the two new quantifiers to the definition of $ETree(\varphi, ext)$. For vertex variables we had to create an successor for every assignment. The same applies for set variables. However there's one big difference. For set variables we only consider assignments over internal vertices. The idea is that the hypergraph where the vertex is internal decides whether or not the vertex is in the set. Combining evaluation tables can then be seen as uniting the sets.

9.1.1 Defining evaluation trees for *MSO*

We add one additional rule to the definition of evaluation trees.

For $\varphi = QS\ \psi$ we define elements $t \in ETree(\varphi, ext)$ as

$$t = (S)$$

As we decide membership only for internal vertices all the successors are anonymous. Also there is no reason to introduce a blank set. Therefore this node only consists of a set of successors.

9.1.2 Generating evaluation trees for *MSO*

Now we modify the evaluation tree generating function accordingly. We have to modify the signature of $tree'$ to fit the assignments for set variables in.

Given a terminal hypergraph $H = (n, E, att, lab, ext)$ and an *MSO*-formula φ we generate the evaluation tree $tree(H, \varphi) \in ETree(\varphi, ext)$ as follows:

$$tree(H, \varphi) = tree'(H, \varphi, \epsilon, \epsilon)$$

If the formula φ is quantifier free ($qr(\varphi) = 0$) we are in the base case.

$$tree'(H, \varphi, \bar{a}, \bar{A}) = evalPredMSO(1, \bar{a}, \bar{A}) \bullet ... \bullet evalPredMSO(pr(\varphi), \bar{a}, \bar{A})$$

As we introduced new predicates we have to redefine our predicate evaluation function.

$$evalPredMSO(i, \bar{a}, \bar{A}) = \begin{cases} true & \text{if } \psi = pred(\varphi, i) = p(\bar{x}) \text{ and } H \vDash \psi(\bar{a}) \\ true & \text{if } \psi = pred(\varphi, i) = S(x) \text{ and } \bar{a}[x] \in \bar{A}[P] \\ false & \text{otherwise} \end{cases}$$

If φ is not quantifier free and $\varphi = Qx\ \psi$)

$$tree'(H, \varphi, \bar{a}, \bar{A}) = (S, s_1, ..., s_{ext}, s_b)$$

where

$$\begin{aligned} s_i &= tree'(H, \varphi, \bar{a} \bullet i, \bar{A}) \quad \text{for } i \in \underline{ext} \\ s_b &= tree'(H, \varphi, \bar{a} \bullet \square, \bar{A}) \end{aligned}$$

and

$$S = \{tree'(H, \varphi, \bar{a} \bullet v) \mid v \in \underline{n} \setminus \underline{ext}\}$$

Otherwise if $\varphi = QS\ \psi$. In this case we have to try every possible subset of internal vertices

$$tree'(H, \varphi, \bar{a}, \bar{A}) = (S)$$

where

$$S = \left\{tree'(H, \varphi, \bar{a}, \bar{A} \bullet M) \mid M \subseteq (\underline{n_H} \setminus \underline{ext})\right\}$$

9.1.3 Interpreting evaluation trees for *MSO*

Interpretation in fact does not need any further explanations. We again add two new cases which are pretty intuitive.

If φ has the form $QS\,\psi$

Case 1: $Q = \forall$

$$evalTree(t) = \begin{cases} false & \text{if } evalTree(s) = false \text{ for an } s \in S \\ true & \text{otherwise} \end{cases}$$

Case 2: $Q = \exists$

$$evalTree(t) = \begin{cases} true & \text{if } evalTree(s) = true \text{ for an } s \in S \\ false & \text{otherwise} \end{cases}$$

9.1.4 Nonterminal evaluation tables

When defining evaluation tables for nonterminal hypergraphs we have to take care. For the terminal case we had only anonymous successors because every internal vertex is anonymous for terminal hypergraphs. However if the hypergraph is nonterminal we have to create nodes with identified successors. Therefore the

definition of $t \in NETree(\varphi, n)$ for $\varphi = QS\ \psi$ is

$$t = (S_{M_1}, ..., S_{M_k}), M_i \subseteq (\underline{n} \setminus \underline{ext})$$

where $k = 2^{n-ext}$ is the number of subsets of internal vertices. Note that we have a set of successors for every subset. When merging with other tables we have to look at every possible combination of sets yielding multiple successors with the same local distribution.

The last thing to do is modifying the *merge* function. Here we again have to only add the new cases

If we have $t_1 \in NETree(\psi, n), t_2 \in ETree(\psi, ext)$ with $\psi = QS\ \omega$. Then

$$\begin{aligned} t_1 &= (S_{M_1}, ..., S_{M_k}), M_i \subseteq (\underline{n} \setminus \underline{ext}) \\ t_2 &= (S) \end{aligned}$$

$$merge(t_1, t_2, \bar{a}) = (S'_{M_1}, ..., S'_{M_k}), M_i \subseteq (\underline{n} \setminus \underline{ext})$$

The single sets have to be defined as every possible combination of sets in t_1 and t_2

$$S'_M = \{merge(S_1, S_2, \bar{a} \mid S_1 \in S_M, S_2 \in S\}$$

At last we have to add one case to $nt2t$

$$nt2t((S_{M_1}, ..., S_{M_k}) = \bigcup_{i=1}^{k} S_{M_k}$$

9.2 Correctness

The correctness of the definitions can be proved by extending the corresponding proofs for *FO* by the necessary cases.

10 Summary

This thesis introduced an algorithm to decide *FO* and *MSO* over languages of hypergraphs. We started with an simple evaluation strategy(tables) and improved it by exploiting properties of both hyperedge replacement grammars and *FO*-formulas. Although the worst case complexity of our algorithm is still bad, there are reasons to assume that in many practical cases the runtime will be good enough to put this algorithm to use. Additionally the theoretic proof of the decidability indicates that the worst case complexity is unavoidable for this kind of problem.

References

[1] Grzegorz Rozenberg Handbook of graph grammars and computing by graph transformation: volume I. foundations 1997

[2] Annegret Habe Hyperedge Replacement: Grammars and Languages 1992

[3] Bruno Courcelle The Monadic Second-Order Logic of Graphs I. Recognizable Sets of Finite Graphs 1990

[4] Annegret Habel and Hendrik Radke Expressiveness of graph conditions with variables

[5] Michael Huth and Mark Ryan Logic in Computer Science: Modelling and Reasoning about Systems 1999

[6] Heinen, Jonathan and Noll, Thomas and Rieger, Stefan Juggrnaut: Graph Grammar Abstraction for Unbounded Heap Structures. 2010