A Translation from Object-Based Hypergraph Grammars into $\pi$-Calculus

Luciana Foss$^{1,2}$ Leila Ribeiro$^{1,3}$

Instituto de Informática
Universidade Federal do Rio Grande do Sul
Porto Alegre, Brazil

Abstract

Object-based models offer abstract constructions to describe complex systems. The Object-Based Graph Grammar (OBGG) is a formalism that may be used to describe this kind of system. This formalism is very intuitive, however, up to now, there are no automatic tools for verification of OBGGs. In this work we propose a translation from Object-Based Hypergraph Grammars into $\pi$-Calculus. So, we may be able to prove properties of the systems modeled in this kind of graph grammars through this translation and automatic checkers for $\pi$-calculus.

Keywords: Graph Grammar, $\pi$-Calculus, Object-Based Systems.

1 Introduction

One of the main aims of rigorous software development is to assure the correctness of the developed system. The basis of a rigorous development is the use of a formal specification method, with syntax and semantics well defined. There are several formalisms for specification of computational systems and the choice of which one to use depends on the characteristics of the application to be developed. Object-based models offer an abstraction level that has been successfully applied in practice, where operations and data are described together within one object. Object-based models are specially well-suited to the

---

1 This work was supported by CNPq, FAPERGS (project ForMOS) and CNPq/CNR.
2 Email: lfoss@inf.ufrgs.br
3 Email: leila@inf.ufrgs.br

1571-0661/© 2004 Elsevier B.V. All rights reserved.
specification of concurrent and cooperating systems. A formal object-based specification language was proposed in [1]: Object-Based Graph Grammars (OBGG). OBGGs are a restricted form of graph grammars [2] that, besides the features of object-based languages, offer a visual specification language, which is usually welcomed by practitioners. However, to be really useful in practice, there should be a (preferably automatic) way to verify whether the desired properties of a system are fulfilled by the model constructed using graph grammars. Up to now, there are no automatic tools for verification of OBGGs. Instead of constructing such tools from scratch, an alternative way is to define a (semantics preserving) translation from this specification language into another for which automatic verification tools already exist. This is what we will do in this paper.

The $\pi$-calculus [8] is a well known and established formalism for description of semantics of concurrent systems. There are some automatic checkers for this formalism, for example, HAL [4] and MWB (Mobility Workbench) [10]. Although one may use the $\pi$-calculus to write specifications, its original intention was to serve as a semantic model language. And indeed, specifications of practical applications written in $\pi$-calculus tend to become large and cumbersome.

In this work we define a translation from OBGG into $\pi$-calculus that is a first step to join the advantages of both methods: the visual, intuitive, and object-based style of graph grammars and the verification tools and semantical model of $\pi$-calculus. Moreover, we prove that this translation preserves the semantics of OBGGs.

In section 2 we present the main definitions of object-based hypergraph grammars (a variation of OBGGs). In section 3 we review the syntax and semantics of $\pi$-calculus. Besides, subsection 3.3 brings a model of agents that characterizes the translation from an OBHG into $\pi$-calculus. In section 4 we present the proposed translation and show that the translation preserves the semantics of graph grammars.

2 Object-Based Hypergraph Grammars - OBHG

Graph grammars offer a natural way to express complex situations. The system states are described by graphs and the dynamic aspects may be captured by grammar rules.

We use, in this work, a model based on Object-Based Graph Grammar (OBGG) [1], called Object-Based Hypergraph Grammars (OBHG). In this model, objects and messages are represented by vertices and hyperedges, respectively. Each hyperedge has one target (the target of the message) and
zero or more sources (the parameters of the message). The internal state of an objects will not be considered, that is, the objects do not have attributes. An extended version of this work including attributes is currently under development.

Object reactions to the receipt of a message will be modeled by grammar rules. Each rule describes the processing of only one message, that may result in the creation of vertices (objects) and/or hyperedges (messages). The hyperedge corresponding to the processed message is deleted with the rule application. We may have more than one rule processing the same kind of message, where the choice of the rule to be applied is non-deterministic, modeling non-deterministic operations. The concurrency between objects and the internal concurrency are modeled by parallel application of rules. The object and message types are described by the type hypergraph of a grammar.

2.1 Syntax

**Note 1** Let $f : A \to B$ be a partial function, then:

- the domain and range of $f$ are denoted, respectively, for $\text{dom}(f)$ and $\text{rng}(f)$;
- the functions $f^*: \text{dom}(f) \to A$ and $f!: \text{dom}(f) \to B$ denote respectively, the domain inclusion and domain restriction;
- the function $f^*: A^* \to B^*$ is the extension of $f$ for lists.

We may have different types of objects and messages in a system. In this work, we use a type hypergraph to specify the type of each system element.

**Definition 2.1** A hypergraph is a tuple $H = (V_H, E_H, sc^H, tg^H)$, where $V_H$ is a set of vertices, $E_H$ is a set of (hyper)edges, $sc^H : E_H \to V_H^*$ is a total function that maps each hyperedge to a list of source vertices and $tg^H : E_H \to V_H$ is a total function that maps each hyperedge to a target vertex. A partial hypergraph morphism $g : G \to H$ is a tuple $g = (g_V, g_E)$ consisting of two partial functions $g_V : V_G \to V_H$ and $g_E : E_G \to E_H$ such that the diagrams (1) and (2) commute, that is, they preserve the source and target functions. A morphism $g$ is total, injective or inclusion if $g_V$ and $g_E$ are total, injective or inclusions, respectively. The category of hypergraph and partial hypergraph morphisms is denoted by $\text{HGraphP}$.

![Diagram](https://via.placeholder.com/150)

Given a hypergraph $T$, a hypergraph typed over $T$ is a tuple $H_T = (H, t^H)$,
where $H$ is a hypergraph and $t^H : H \to T$ is a total hypergraph morphism. A typed hypergraph morphism $g : H^T \to G^T$ between typed hypergraphs $H^T$ and $G^T$ is a hypergraph morphism $H \to G$, such that the diagram (3) commutes in $H\text{Graph}P$. The category of typed hypergraphs and typed hypergraph morphisms is denoted by $TH\text{Graph}P(T)$.

In a hypergraph grammar, the states changes are described by rules. This rules, called OBHG rules, may be applied to a hypergraph (state of system) and change it. An OBHG rule should have only one hyperedge on its left-hand side, that should be consumed when it is applied, and only source and target vertices should appear on its left-hand side.

**Definition 2.2** Let $T$ be a hypergraph. Then an OBHG rule with respect to $T$ is an injective typed hypergraph morphism $r : L^T \to R^T$ in $TH\text{Graph}P(T)$, such that:

(a) $|E_L| = 1$;
(b) $\forall a \in E_L. a \notin \text{dom}(r)$;
(c) $\forall x \in V_L. x \in \text{dom}(r)$;
(d) $\forall x \in V_L. \exists a \in E_L. t_L(a) = x \vee x \in \text{sc}_L(a)$.

The class of all OBHG rules with respect to $T$ is denoted by $HRules(T)$.

**Definition 2.3** A typed Object-Based Hypergraph Grammar (OBHG) is a tuple $HG = (T, I, N, n)$, where:

- $T$ is a finite hypergraph, called type hypergraph (type of grammar);
- $I$ is a typed hypergraph in $TH\text{Graph}P(T)$ (initial hypergraph of grammar);
- $N$ is a set of rule names;
- $n : N \to HRules(T)$ is a total function that associates each rule name to a rule.

**Example 2.4** In this example we show an OBHG specification of an object-based system. The TYPE HYPERGRAPH $T$ (Fig. 1(a)) identifies the object and message types. In this example, there are three types of objects: circle, star and square; and four types of messages: ope1, ope2, ope3 and ope4. The INITIAL HYPERGRAPH $H$ (Fig. 1(a)) specifies the initial state of object-based system, where there are one instance of object circle, one instance of
object \textit{star}, two instances of object \textit{square}, one instance of message \textit{ope}1, one instance of message \textit{ope}2 and two instances of message \textit{ope}3. The rule names \{r1, r2, r3, r4\} are rule identifiers that describe the object operations. The \textsc{naming function} (Fig. 1(b)) associates the names to rules.

2.2 Semantics

The operational semantic of a hypergraph grammar is defined in terms of derivation steps, that are applications of the rules of the grammar to some state. The result of application of rule \( r : L \rightarrow R \) to a hypergraph \( I \) is obtained in the following way:

(i) Add to \( I \) all items created by rule \( r \);

(ii) Delete from resulting hypergraph of the step 1, all items deleted by rule \( r \);

(iii) Delete pendent hyperedges, that is, hyperedges links to vertices deleted in 2 step.

Formally, a derivation step is given by a pushout in the category \textsc{THGraphP}(T) \cite{5}. A rule may be applied to a hypergraph if there is an occurrence of its left-hand side in the hypergraph. This occurrence is described by a total typed hypergraph morphism, called match. If a rule \( r \) is applicable to \( G^T \) by a match \( m \) and results in a hypergraph \( H^T \), then a derivation step \( G^T \xrightarrow{r,m} H^T \) is obtained. The pushout is unique up to isomorphism, so \( H^T \) represents a set of isomorphic objects.

\textbf{Definition 2.5} Let \( r : L \rightarrow R \) be a rule and \( IN \) be a typed hypergraph. Then a \textbf{match} is a total typed hypergraph morphism \( m : L \rightarrow IN \). A \textbf{derivation step} \( s \), with name \( nr \), of a hypergraph \( IN \) with rule \( r \) at match \( m \), is tuple \( s = (nr, S) \), where \( S \) is the pushout of \( m \) and \( r \) in \textsc{THGraphP}(T) and \( n(nr) = r \). In this case, we write \( IN \xrightarrow{nr,m} FI \). This step derivation is given by the diagram bellow, where \( IN, FI, r' \) and \( m' \), are called initial and final hypergraph, co-rule and co-match, respectively. The class of all derivation steps using the rules of a hypergraph grammar \( HG \) is denoted by \( \text{Step}_{HG} \).

\begin{center}
\begin{tikzpicture}

\node (L) at (0,0) {$L$};
\node (R) at (2,0) {$R$};
\node (IN) at (0,-1) {$IN$};
\node (FI) at (2,-1) {$FI$};
\node (S) at (1,0) {$S$};

\draw[->] (L) -- (R) node[above] {$r$};
\draw[->] (IN) -- (S) node[below] {$m$};
\draw[->] (S) -- (FI) node[below] {$m'$};
\end{tikzpicture}
\end{center}

In order to compare the semantics of the OBHG with the one of \( \pi \)-calculus, we define the interleaving semantics for OBHG, because this is the usual semantic model of \( \pi \)-calculus. In this model, the non-determinism is described by different sequences of derivations with common subsequence. The con-
currency is described by different derivation steps observed in two different orders.
The interleaving semantics is defined by a set of grammar computations. A derivation step describes only one step of a computation using a hypergraph grammar. Whole computations may be described by sequences of derivation steps in which the output hypergraph of a step is the input hypergraph of the next step.

**Note 2** If \( C \) is a set (or class), then the set (or class) of all sequences over \( C \) is denoted by \( C^\infty \) (for finite sequences: \( C^* \)). The empty sequence is denoted by \( \lambda \). If \( \sigma \in C^\infty \), then \( |\sigma| \in \mathbb{N} \cup \{\omega\} \) is the length of \( \sigma \). The \( i^{th} \) element of a sequence \( \sigma \) is denoted by \( \sigma_i \).

**Definition 2.6** The class of sequential derivations with respect to \( HG = \langle T, I, N, n \rangle \) is defined by:

\[
SDer_{HG} = \{ \sigma \in \text{Step}_{HG}^\infty \mid \sigma = \lambda \lor IN_{\sigma} = IN_{\sigma_1} = I \land FI_{\sigma_i} = IN_{\sigma_{i+1}} \text{ for } 1 \leq i < |\sigma| \}
\]

We use \( s \in SDer_{HG} \) if exists \( \sigma \in SDer_{HG} \) and if \( s \) is a step of sequence derivation \( \sigma \). The set of all states \( IN \) and \( FI \) of all sequence derivations in \( SDer_{HG} \) is called \( \text{State}_{HG} \), in other words, \( \text{State}_{HG} = \{ G \mid H \in SDer_{HG} \lor H \rightarrow R T \} \).

The operational semantics of a hypergraph grammar may be described by a Labeled Transition System (LTS). A LTS is defined by the following components: a set of system states, a set of transition labels, the initial state and a transition relation.

**Definition 2.7** Given an OBHG \( HG = \langle T, I_{HG}, N, n \rangle \), the OBHG semantics of \( HG \), denoted by \( \text{SemOBHG}(HG) \), is defined by LTS \( ST = (S, R, I, \rightarrow) \), where:
- \( S = \text{State}_{HG} \);
- \( R = \{ nr.t(id) \mid id \in E_G \land G \in S \land n(nr) = L^T \rightarrow R^T \land t = t^G(id) \land e \in E_L \} \), where \( t = t^G(id) \);
- \( I = I_{HG} \);
- \( \rightarrow \) is given by following rule:

\[
\begin{align*}
H \xrightarrow{nr,m} H' & \in SDer_{HG} \\
H \xrightarrow{nr,T(id)} H' & \quad t = t^G(id) \land id = m(msg) \land msg \in A_L \land \\\n\quad & n(nr) = L^T \rightarrow R^T
\end{align*}
\]
3 \(\pi\)-Calculus

The \(\pi\)-calculus [8] is a process algebra that handles channels as messages, thus modeling processes that may have changing structure. The basic computational step is the sending of a channel between two processes. The receiver process may use the new channel in future interactions. The channels in the \(\pi\)-calculus are only names, atomic entities without internal structure.

3.1 Syntax

The basic syntax of \(\pi\)-calculus can be seen in Fig. 2. Here, we assume an infinite set of names \(N\), ranged over by \(a, b, \ldots\) and a set of agents ranged over by \(P, Q, \ldots\) A term of \(\pi\)-calculus is given by an agent.

\[
\begin{align*}
\text{Agents} & & P ::= 0 & \text{Null} \\
& & \alpha.P & \text{Prefix} \\
& & P + P & \text{Sum} \\
& & P | P & \text{Parallel} \\
& & (\nu x)P & \text{Restriction} \\
& & [x = y]P & \text{Match} \\
& & !P & \text{Replication} \\
& & A(y_1, \ldots, y_n) & \text{Identifier} \\
\end{align*}
\]

\[
\begin{align*}
\text{Prefixes} & & \alpha ::= \pi x & \text{Output} \\
& & a(x) & \text{Input} \\
& & \tau & \text{Silent} \\
\end{align*}
\]

\[
\begin{align*}
\text{Definitions} & & A(x_1, \ldots, x_n) \overset{\text{def}}{=} P \quad (\text{where } i \neq j \implies x_i \neq x_j) \\
\end{align*}
\]

Figure 2. The \(\pi\)-calculus syntax

The input Prefix and the Restriction are operators that bind names. For example, in \(a(x).P\) and \((\nu x)P\) the occurrences of \(x\) are said to be bound in \(P\), that is, the scope of \(x\) is \(P\). An occurrence of \(x\) is said to be free if it is not bound. The set of bound names of \(P\) is denoted by \(bn(P)\) and the set of free names is denoted by \(fn(P)\). In a Definition \(A(x_1, \ldots, x_n) \overset{\text{def}}{=} P\) we assume that \(fn(P) \subseteq \{x_1, \ldots, x_n\}\).

Two processes \(P\) and \(Q\) are \(\alpha\)-convertible if \(Q\) can be obtained from \(P\) by a finite sequence of substitutions of bound names. A substitution is a function from names to names. We write \(\{x/y\}\) for the substitution that maps \(y\) to \(x\) and is the identity for all other names.

Note 3

- The Sum \(P_1 + \cdots + P_n\) is written as \(\sum_{i=1}^{n} P_i\). If \(n = 0\), Sum is equivalent to 0.
- A restriction sequence \((\nu x_1) \cdots (\nu x_n) P\) is written as \((\nu x_1, \cdots, x_n) P\).
- The parallel composition \(P_1 | \cdots | P_n\) is written as \(\prod_{i=1}^{n} P_i\). If \(n = 0\), this composition is equivalent to 0.
- The action sequences \(a(x_1). \cdots a(x_n)\) and \(\overline{a} x_1. \cdots \overline{a} x_n\) are written as \(\bigcirc_{i=1}^{n} a(x_i)\) and \(\bigcirc_{i=1}^{n} \overline{a} x_i\), respectively. If \(n = 0\), this term is 0.
- In a Prefix we can suppress the object if it is not essential, for example, \(a.P\) represent the agent \(a(x).P\) where \(x\) is never used.
- We can suppress the null 0 if there is not confusion. For example, we can write \(\alpha\) for the agent \(\alpha.0\).

A system specification should describe the initial state of this system. In \(\pi\)-calculus, this is done through a term. In this specification, all agent identifiers must be defined.

**Definition 3.1** A \(\pi\)-calculus specification \(SP\) is given by tuple \((T, D)\), where \(T\) is a \(\pi\)-calculus term that describe the initial state of system, called initial term, and \(D\) is the set of definitions of agents. All definitions of agents in \(T\) must be in \(D\).

### 3.2 Semantics

The operational semantics of a process algebra is usually given by LTS. The \(\pi\)-calculus follows this pattern and most of the transition rules are similar to those of the other algebras. The semantics of a specification \(SP = (T, D)\), denoted by \(SemPi(SP)\) is defined as following:

- The system states are given by terms of the \(\pi\)-calculus;
- The transition labels are given by actions performed by agents: \(\tau, \overline{xy}, x(y)\) and \(\overline{a}(y)\);
- The initial state of the LTS is the initial term of the \(\pi\)-calculus specification;
- The transition relation is given by the set of rules showed in Fig. 3.

### 3.3 Object-Based Model described in \(\pi\)-Calculus - OBM-\(\pi\)

The translation proposed in this work is the translation from objects and messages and their relationships into agents of the \(\pi\)-calculus. These target agents have specific forms. In this section we will define the kind of \(\pi\)-calculus terms that characterize the translated graph grammars. This characterization, called Object-Based Model described in \(\pi\)-Calculus, short OBM-\(\pi\), will be very useful to aid the definitions and to carry out the proof in the next sections.

In an OBM-\(\pi\), the objects and messages are defined as processes of the \(\pi\)-
calculus, called object agents and message agents, respectively. These agents communicate through a local channel. The source object and the parameters of each message are represented as parameters of message agents.

The reactions of the objects when receiving a message are described by rule agents, that compose the object agent. These rule agents are identified by kind of treated message and by a “rule” identifier. A “rule” describes the procedures to treat a message. Each kind of message can have several procedures for treating it, so we can have several rule agents that describe the treatment of the same kind of message. The choice of procedure to be executed is non-deterministic. This is described by composition of rule agents (for a same kind of message) with the Sum operator (+), without guard.

The concurrency between objects is modeled by parallel composition of object agents and message agents. So each object can treat its messages in parallel. The internal concurrency is modeled by recursion of object agent.

This model is described by a specification in π-calculus ($I_\pi, A_\pi$). $I_\pi$ (1) is the term that represent the initial state of system. This term is composed by parallel composition of agents that represent the objects and messages (defined in $A_\pi$). The message and object identifiers of $I_\pi$ are treated as a local channel. $A_\pi$ is a set of system agents (objects and messages) definitions.

**Note 4** Let $C$ be a set and $l$ be a list, then:
- $|C|$ is the number of elements in the set $C$ and $|l|$ is the number of elements of $l$;
- $e \in l$ means that $e$ belongs to $l$ and $l \subseteq C$ means that all elements of $l$ belongs to $C$;
- The $i^{th}$ element of $l$ is denoted by $l[i]$.
- $\tilde{p}$ shortens $p_1, \ldots, p_n$, where $n = |p|$;
- $\tilde{p}_i$ shortens $p_{1,i}, \ldots, p_{n,i}$, where $n = |p|$;
- The elements of set $A_\pi$ can be shortened to $\oplus(t, id, d, tp)$ and $O_{to}(id)$.
- A subterm $(A_1 \cdots A_n)$ of $P$ is denoted by $a_P$ and we use $A_i \in a_P$ to express that there is an agent $A_i$ in this subterm.

**Definition 3.2** Given the finite sets $TO, TM$ and $P$ (set of object types, set of message types and set of number of message parameters, respectively), the set $RM_{tm}$ of rule identifiers that treat messages with type $tm$, and the set $TM_{to} \subseteq TM$ that contain the types of messages treated by object with type $to$, an **Object-Based Model described in $\pi$-calculus** (OBM-$\pi$) is a $\pi$-calculus specification $PM = (I_\pi, A_\pi)$, where:

1. $I_\pi = (vid_1, \ldots, id_n) a_{I_\pi}$

   where $a_{I_\pi} = (A_1 \cdots A_n)$, $id_i$ is an identifier such that $id_i = id_j \Rightarrow i = j$ and $A_i = O_t(id_i)$ or $A_i = \oplus(t, id_i, d, v[1], \ldots, v[m])$, with $v, d \in \{id|O_t(id) \in a_{I_\pi}\}$, $0 < k \leq m$ and $m \geq 0$. If $m = 0$, then $A_i = \oplus(t, id_i, d)$.

2. $A_\pi = Ms \cup Ob$

3. $Ms = \{ag\_msg_t \mid t \in TM\}$, where

4. $ag\_msg_t = \oplus(t, id, d, id, \tilde{tp}) = \{\forall m\mid m.t \supseteq id, \forall i \in \{1, \ldots, |p|\}, \forall (id, \tilde{tp}) \in p, t \in TO \text{ and } |p| \in P.$

5. $Ob = \{ag\_obj_{to} \mid to \in TO\}$, where

6. $ag\_obj_{to} = O_{to}(id) = \{\forall m\mid m.t \supseteq id, \sum_{t \in TM_{to}} [t = tm]M_{tm}(m, id, t) \mid O_{to}(id)),$

7. $M_{tm}(m, d, t) \overset{def}{=} \sum_{nr \in RM_{tm}} R_{tm.nr}(m, d, t, nr)$,

8. $R_{tm.nr}(m, d, t, nr) \overset{def}{=} \sum_{k=1}^{n} m(p_k).((vid_1, \ldots, id_s, msg_1, \ldots, msg_j))$

   $(\prod_{i=1}^{s} CriaObj(id_i, to_i)) \mid (\prod_{i=1}^{s} id_i).m(id).\overrightarrow{7}id.$
\[
(\prod_{i=1}^j \ominus(t_i, msg_i, d_i, v_i[1], \ldots, v_i[a_i]))\),
\]

\[n, a \in P, s, j \in \mathbb{N},\]

(9) \[
CriaObj(id, t) \overset{def}{=} \sum_{c \in TO} ([t = c\overrightarrow{id}.O_c(id)).
\]

\(A_\pi(2)\) is formed by union of sets \(Ms\) and \(Ob\).

\(Ms(3)\) contain the definitions of message agents. For each kind of message, there is one element in this set. A message agent (4) has a type \(t\), an identifier \(id\), a target \(d_id\) and a parameters list \(\hat{p}_{tp}\), where \(td\) and \(tp\) identify the type of the target object and type of message parameters, respectively. The number of parameters of message is given by length of \(\hat{p}\).

Each message agent has a local channel \(m\) that will be sending to target object agent through the channel \(d_id\) (target object identifier of the message). So, the channel \(m\) will be local for two agents (message and target object). Afterwards the message agent sends to the target object, through \(m\), its type, its parameters and its identifier (\(id\)). Finally it terminates.

The set \(Ob(5)\) contains the definitions of object agents. For each kind \((to \in TO)\) of object there is one element in this set. Each object (6) has an identifier \(id\), through which it receives the channel \(m\). This channel is used for receiving a message. This agent has a parallel composition that allows the parallel treatment of different messages by one object. This occurs because after synchronization with a message, the agent presents its previous behavior in parallel to the behavior of a subterm that will treat the received message.

The object \(Ob\) receives through channel \(m\) the kind of message that it will treat. Each object \(to\) has (in a sum) one agent \(M_{tm}\) for each \(tm \in TM_{to}\), where \(TM_{to}\) is a set of message types that the object \(to\) can treat. The agent \(M_{tm}\) that treats the received message type is selected by the match \([t = tm]\) that guards it.

The agent \(M_{tm}(7)\) is composed by a sum of agents \(R_{tm,nr}\). This sum represents the different actions that can be taken to treat the message type \(tm\). Each \(nr \in RM_{tm}\) (where \(RM_{tm}\) is a set of names of rules that describe the treatment of message type \(tm\)) represents one different treatment for a same message type, allowing to represent the non-determinism.

The agent \(R_{tm,nr}(8)\) describes the treatment of messages. Initially this agent receives the identifiers of message parameters. Afterwards, new objects and/or messages are created and their identifiers are declared as local names.

The objects of a system are created by agent \(CriaObj(9)\), which contains all object types \(c \in TO\) of a system. The created object type is selected through its type \(([t = c])\).

All objects must be created before messages. This is assured by guard
(\prod_{i=1}^{*} id_i) that allows continuation only after synchronization with all created objects. Afterwards, the object receives the identifier (id) of the message being treated, synchronizes with environment through the channel \(nr\) and sends the id to the environment through the channel \(t\). These two last actions signalize what message (and its type) was treated and what rule was applied.

Finally, the messages are created. Each created message \(i\) has an identifier \((msg_i)\), a type \((t_i)\), a target \((d_i)\) and a parameters list \((v_i)\). The target and parameters of created messages can be either the object that is treating the message \((d)\), or some parameter of treated message \((p)\), or some created object \((id)\).

In the transition system of an OBM-\(\pi\) there are some states that represent the complete application of rules, that is, all steps of a rule that began to be applied were accomplished. These terms are called Complete Terms.

Definition 3.3 Given an OBM-\(\pi\) \(PM = (I_\pi, A_\pi)\), a complete term \(T_\pi\) of \(PM\) is a term of \(\pi\)-calculus such that:

i) there is a path \(I_\pi \xrightarrow{a^*} T_\pi\) in \(SemPi(PM)\);

ii) \(T_\pi\) has the pattern: \(T_\pi = (\nu id_1, \ldots, id_n) a_{T_\pi}\), where \(a_{T_\pi} = (A_1| \cdots | A_n)\) and \(id_i\) is an identifier such that \(A_i = (\nu id_{n+1}, \ldots, id_{n+1+o}) a_{T_\pi}'\) or \(A_i = Ot(id_i)\) or \(A_i = \@(t, id_i, d, v_1, \ldots, v_m)\), with \(v_k, d \in \{id|Ot(id) \in a_{T_\pi}\}\), \(0 < k \leq m\) and \(m \geq 0\). If \(m = 0\), then \(A_i = \@(t, id_i, d)\), and \(a_{T_\pi}'\) is a complete term.

The set of all complete terms of an OBM-\(\pi\) \(PM\) is called Termo-\(CO_PM\).

In a complete term, there may be repeated names of bound variables, but with different scopes. A term structurally congruent to a complete term that does not have repeated bound variable names is called OB Term.

Definition 3.4 Given an OBM-\(\pi\) \(PM = (I_\pi, A_\pi)\), an OB term \(B\) of \(PM\) is a complete term of \(PM\) where:

i) \(B = (\nu id_1, \ldots, id_n) a_B, a_B = (A_1| \cdots | A_n)\) and \(A_i = (\nu id_{n+1}, \ldots, id_{n+1+o}) a_B'\) or \(A_i = Ot(id_i)\) or \(A_i = \@(t, id_i, d, v_1, \ldots, v_m)\), with \(v_k, d \in \{id|Ot(id) \in a_B\}\), \(0 < k \leq m\) and \(m \geq 0\). If \(m = 0\), then \(A_i = \@(t, id_i, d)\), \(a_B'\) is a complete term and;

ii) \(\forall id_i \in C.id_i = id_j \Rightarrow i = j\).

The set of all OB terms of an OBM-\(\pi\) \(PM\) is called TermoOB\(_{PM}\).

The message and object types of an OBM-\(\pi\) can be described by a hypergraph. This hypergraph is obtained from \(A_\pi\).

Definition 3.5 Given an OBM-\(\pi\) \(PM = (I_\pi, A_\pi)\), the type of this model is a hypergraph \(HG_{PM} = (V,E,sc,tg)\), where
\[- \quad V = \{to \ | \ O_{to}(id) \in A_\pi\};
\]
\[- \quad E = \{t \ | \ \@\!(t, id, d_{id}, \tilde{p}_t) \in A_\pi\};
\]
\[- \quad \forall a \in E, \ tg(a) = td, \text{ such that } \@\!(a, id, d_{id}, \tilde{p}_t) \in A_\pi \text{ or } \@\!(a, id, d_{id}) \in A_\pi; \]
\[- \quad \forall a \in E, \ sc(a) = \begin{cases} tp & \text{if } \@\!(a, id, d_{id}, \tilde{p}_t) \in A_\pi \\ \{\} & \text{if } \@\!(a, id, d_{id}) \in A_\pi \end{cases} \]

4 Translation from OBHG into OBM-\(\pi\)

An OBM-\(\pi\) can be obtained from an OBHG specification through a translation. Now we will define the function performs this translation. The defined terms are not terms of \(\pi\)-calculus, but formulae to obtain them. The terms are instances of these formulae. Each formula has variables that assume different values in different translations. These variables (in bold) should be instantiated at the moment of translation. In some cases, there are functions (instead of variables) that must be evaluated at the moment of translation.

**Note 5** If \(l\) is a list, the notation \((\nu l[1], \ldots, l[|l|])\) is shortened to \((\nu l)\).

**Definition 4.1** Given an OBHG \(HG = (T, I_T, N, n)\), with \(T = (V_T, E_T, sc_T, t^T)\) and \(I = ((V_I, E_I, sc_I, t^I), t^I, T)\), a translation of \(HG\) into an OBM-\(\pi\) is defined by \(Trad_{HG}(HG) = (I_\pi, A_\pi)\), where: \(I_\pi = Trad_H(I)\), \(A_\pi = Trad_R(T, N, n)\) and

\[
\begin{align*}
(10) \quad Trad_H(I) & = (\nu l_v, la)\left(\prod_{i=1}^{|l_v|} O_{t^I(l_v[i])}(l_v[i]) \prod_{|la|} \prod_{j=1}^{|la|} \@\!(t^I(la[j]), la[j], t^I(la[j]), p_j[1], \ldots, p_j[n_j]))\right),
\end{align*}
\]

where \(l_v = Lista(V_I), la = Lista(E_I), p_j = sc^I(la[j]), n_j = |p_j|\)

\[
(11) \quad Ms = \{@\!(t, id, d_{id}, \tilde{p}_t) \mid t \in E_T, d_{id} = t^T, t = sc^T(t)\}, \quad n = |t| \quad where \quad t \in E_T, d_{id} = t^T, t = sc^T(t)\]

\[
(12) \quad Ob = \{O_{\nu}(id) \mid id(m) \cdot m(t)\}, \quad \sum_{a \in DestMsg(v, T)} [t = a] M_a(m, id, t)\}
\]

where
Each of these identifiers is declared as a local channel (\((\nu \text{lv}, \text{lm})\)). The same occurs with the messages, in the second product. There is one instance of message agents for each message in the initial hypergraph \(HG\). In the initial state \((10)\), the lists \(\text{la} \) and \(\text{lv} \) are lists of edge (message) and vertex (object) identifiers, respectively, present in initial hypergraph \(HG\). Each of these identifiers is declared as a local channel \((\nu \text{lv}, \text{la})\).

The first product \((10)\) represents a parallel composition of all objects (all elements of \(\text{lv}\)) in \(HG\). There is one agent definition \(O_{\text{lv}[i]}\) for each object type \((\text{t}^I(\text{lv}[i]))\). The same occurs with the messages, in the second product. There is one instance of message agents for each message in the initial hyper-

\[
M_a(m, d, t) \overset{\text{def}}{=} \sum_{\text{nr} \in \text{RegrasMsg}(a, HG)} R_{a, \text{nr}}(m, d, t, \text{nr}) \land \\
R_{a, \text{nr}}(m, d, t, \text{nr}) \overset{\text{def}}{=} \bigotimes_{i=1}^{\text{se}^T(a)} m(p_i).((\nu \text{lv}, \text{lm}) \\
\prod_{i=1}^{\text{lv}} (\text{CriaObj}(\text{lv}[i], \text{t}^R(\text{lv}[i]))) \prod_{i=1}^{\text{lm}} \text{lv}[i].m(id).\text{nr}.\text{tg}.d \\
\prod_{i=1}^{\text{pm}[i]} (@(\text{t}^R(\text{lm}[i]), \text{lm}[i], \text{dm}[i], \text{pm}[i][1], \cdots, \text{pm}[i][\text{lpR}_i])) \land \\
C\text{riaObj}(id, t) \overset{\text{def}}{=} \sum_{b \in V_T} ([t = b]id.\text{O}_b(id)) \land \\
dm[i] = \begin{cases} 
\text{d} & \text{if } \text{tg}^R(\text{lm}[i]) \in \text{rng}(r) \text{ and } \\
\text{pm}_x & \text{if } \text{tg}^R(\text{lm}[i]) \in \text{rng}(r) \text{ and } \text{sc}^L(\text{trigger}(r)), \\
\text{tg}^R(\text{lm}[i]) & \text{if } \text{tg}^R(\text{lm}[i]) \notin \text{rng}(r) 
\end{cases} \\
\text{lpR}_i[j_i] = \begin{cases} 
\text{d} & \text{if } \text{lpR}_i[j_i] \in \text{rng}(r) \text{ and } \\
\text{pm}_x & \text{if } \text{lpR}_i[j_i] \in \text{rng}(r) \text{ and } \text{sc}^L(\text{trigger}(r)), \\
\text{lpR}_i[j_i] & \text{if } \text{lpR}_i[j_i] \notin \text{rng}(r) 
\end{cases} \\
\text{v} \in V_T \land r = n(\text{nr}) \land \text{lv} = \text{Lista}(\text{NewObj}(r)) \land \\
\text{lm} = \text{Lista}(\text{NewMsg}(r)) \land \text{lpR}_i = \text{sc}^R(\text{lm}[i]) \land 0 < j_i \leq |\text{lpR}_i|
The parameters of these agents are: the type of message \( t^I(la[j]) \); its identifier \( la[j] \); the target object of the message \( tg^I(la[j]) \); and a list of message parameters \( p_j \). The length of list \( p_j \) is the amount of message parameters.

In the set \( Ms(11) \) there is one different agent definition for each edge of the type hypergraph \( T (t \in E_T) \), that identifies the message types of system. The type \( td \) of target vertex of each message is identified in the parameter \( d_{td} \). The list of type of source vertices \( tp \), that are identified in \( p_t.tp[i] \), can be obtained analogously.

In (12), the set of object agents is defined. There is one different definition for each vertex in type hypergraph \( T \). The definitions of agents \( O_v(id) \) are differentiated by their types \( v \). This definition is instantiated changing all names \( a \) by the identifier of each edge that arrives in vertex \( v \in T \). So, there will be as many definitions \( M_a \) as there are edges in \( T \).

In (13) the formula to obtain the definitions of agents \( M_a \) is described. This agent is instantiated changing each name \( nr \), by a rule name in \( N \), which has an edge of type \( a \) in its left-hand side. So, there will be, in sum of this formula, as many agents \( R_{a.nr} \) as there are rules in \( N \).

To instantiate formula (14) of the rule agent definitions, we must change all occurrences of the variable \( lv \) by list of vertices created by \( nr \). In the same way occurrences of the variable \( lm \) must be changed by the list of edges created by \( nr \). Moreover, there are some functions that must be evaluated: \( t^R(lv[i]) \), that is the type of \( i \)th vertex created by \( nr \) and \( t^R(lm[i]) \), that is the type of \( i \)th created edge. The target vertex \( (dm_i) \) of messages created by a rule is evaluated in (16). This vertex can be the target vertex \( (d) \) of the edge deleted by this rule, or one of its source vertices \( (p_x) \), or one of the vertices \( (tg^R(lm[i])) \) created by this rule. The source edges of created edges are evaluated (17) in the same way that the target vertex.

In (15) the definition of the agent responsible by creation of objects is described. To instantiate this formula we must change all occurrences of \( b \) by identifiers of vertices of the type hypergraph \( T \).

In (18) the variables used in the others formulae are defined, where the variable \( lpR_i \) represent the list of source vertices of the \( i \)th edge created by rule \( nr \).

**Example 4.2** Fig. 4 shows the translation from the OBHG of example 2.4 (page 4) into an OBM-\( \pi \).

### 4.1 Semantic Compatibility

The synchronizations performed by agents simulate the grammar rule applications. But these agents perform various silent synchronizations (\( \tau \)) to simulate
Given a GHBO \( GH = (T, H, N, n) \), the translation from GH into its corresponding MBO-\( \pi \) is given by tuple \( (I_\pi, A_\pi) \), where
\[
I_\pi = \langle \nu c_1, q_1, q_2, e_1, m_1, m_2, m_3, m_4 \rangle (O_{circ}(c_1)|O_{quad}(q_1)|O_{quad}(q_2)|O_{estr}(e_1))
\]
\[
\oplus (\nu e_1, m_3, c_1) \oplus (\nu e_2, m_4, e_1, c_1, q_1) \oplus (\nu e_3, m_1, q_1, q_2) \oplus (\nu e_3, m_2, q_1, q_1)
\]
\[
A_\pi = Ms \cup Ob, \text{ where}
\]
\[
Ms = \{ \oplus (\nu e_1, id, d_{circ}) = (\nu m) \overline{d}_{circ} m.\overline{m}_{ope} 1.\overline{id},
\oplus (\nu e_2, id, d_{estr}, p_{1,circ}, p_{2,quad}) = (\nu m) \overline{d}_{estr} m.\overline{m}_{ope} 2.\overline{m}_{p_{1,circ}}.\overline{m}_{p_{2,quad}}.\overline{id},
\oplus (\nu e_4, id, d_{quad}, p_{1,circ}) = (\nu m) \overline{d}_{quad} m.\overline{m}_{ope} 4.\overline{m}_{p_{1,circ}}.\overline{id}\}
\]
\[
Ob = \{ O_{circ}(id) = id(m).\langle m.\langle t = ope 1 \rangle M_{ope 1}(m, id, t) \rangle | O_{circ}(id) \},
\oplus O_{estr}(id) = id(m).\langle t = ope 2 \rangle M_{ope 2}(m, id, t) \rangle | O_{estr}(id) \},
\oplus O_{quad}(id) = id(m).\langle t = ope 3 \rangle M_{ope 3}(m, id, t) +
\oplus [ t = ope 4 \rangle M_{ope 4}(m, id, t) \rangle | O_{quad}(id) \}, \text{ where}
\]
\[
M_{ope 1}(m, d, t) = R_{ope 1.r_1}(m, d, t, r_1)
\]
\[
M_{ope 2}(m, d, t) = R_{ope 2.r_2}(m, d, t, r_2)
\]
\[
M_{ope 3}(m, d, t) = R_{ope 3.r_3}(m, d, t, r_3)
\]
\[
M_{ope 4}(m, d, t) = R_{ope 4.r_4}(m, d, t, r_4)
\]
\[
R_{ope 1.r_1}(m, d, t, nr) = m(id).\overline{m}.\overline{t}.\overline{id}
\]
\[
R_{ope 2.r_2}(m, d, t, nr) = m(p_1).m(p_2).\langle (\nu q_2, m_1) | (CriaObj(q_2, quad)| q_2.m(id).\overline{m}.\overline{t}.\overline{id}.\oplus (\nu p_4, m_1, q_2, p_1) \rangle \rangle
\]
\[
R_{ope 3.r_3}(m, d, t, nr) = m(p_1).\langle (\nu q_3) | (CriaObj(q_3, quad)| q_3.m(id).\overline{m}.\overline{t}.\overline{id} \rangle \rangle
\]
\[
R_{ope 4.r_4}(m, d, t, nr) = m(p_1).\langle (\nu c_2, m_1) | (CriaObj(c_2, circ)| c_2.m(id).\overline{m}.\overline{t}.\overline{id}.\oplus (\nu p_1, m_1, c_2) \rangle \rangle
\]
\[
CriaObj(id, t) =\begin{cases} 
\langle t = circ | id, O_{circ}(id) + \\
\langle t = estr | id, O_{estr}(id) + \\
\langle t = quad | id, O_{quad}(id) \rangle 
\end{cases}
\]

Figure 4. Translation from an OBHG into an OBM-\( \pi \)

the application of only one rule of OBHG. To compare the OBHG and OBM-\( \pi \) semantics we compare the paths created by respective LTS’s. A path is a list of transitions where the final state of one is the initial state of another.

**Definition 4.3** Given a LTS \( ST = (S, R, I, \rightarrow) \), the set of paths of \( ST \) denoted by Path(ST) is defined by: \( C = \{ l | l \in (S \times R \times S)^* \land \forall t_j = (i_j, r_j, p_j), t_{j+1} = (i_{j+1}, r_{j+1}, p_{j+1}) \in l, p_j = i_{j+1} \land j \in \{1..|l| \} \land i_1 = I \} \)

The treatment of one message, in an OBHG, is done applying only one rule,
thus the OBHG transition system presents only one transition for each treated message. The labels of these transitions has the form \( \pi \tau (id) \), that indicates what message \((id)\) was treated and what rule \((nr)\) was applied. On the other hand, in an OBM-\(\pi\), the treatment of one message requires various synchronizations between object and message agents. Thus, the OBM-\(\pi\) transition system has a sequence of transitions for each message. The labels of these transitions are a series of \(\tau s\), followed by the labels that identify the applied rule \((\pi\tau, \text{free output action})\) and the treated message \((\tau(id), \text{bound output action})\). In an OBM-\(\pi\), the resulting synchronizations of parallel treatments of two messages create some paths where the transitions (of treatment of each message) may appear interleaved. In Fig. 5 the left side diagram represents the LTS of an OBHG and the right side diagram represents the LTS of the corresponding translation. The parallelism, in the two LTS’s is represented by existence of two paths where the treatment of messages \(nr \pi T (m1)\) and \(nr \tau T (m2)\) appear in different orders.

\[
\begin{align*}
G_1 & \xrightarrow{nr \pi T (m1)} G_2 \\
G_3 & \xrightarrow{nr \tau T (m2)} G_4 \\
G_2 & \xrightarrow{nr \tau T (m2)} G_3 \\
G_4 & \xrightarrow{nr \pi T (m1)} G_1
\end{align*}
\]

Figure 5. Example of OBHG LTS (left) and of LTS of its translation (right).

In an OBHG the treatment of messages is atomic. In the LTS of the translation of this OBHG there exist paths that represent these atomic treatments. In the diagram of the right side of Fig. 5, these paths are represented by continuous arrows. What characterizes these paths are the labels of its transitions and the final term. These paths are called complete paths. The others paths (represent by dashed arrows) do not represent existent behavior in OBHG. So, the semantic compatibility is given only for paths in OBM-\(\pi\) LTS that represent the atomic treatment of messages, that is, for all complete paths. This choice creates questions that will be discussed in the conclusions.

**Definition 4.4** Given an OBM-\(\pi\) \(PM\), a path \(c \in \text{Path}(\text{SemPi}(PM))\) is said to be complete if \(\forall t = (i, l, p) \in c\), where \(l\) is a bound output action,
\( p \in \text{TermoCO}_{PM} \). The set of all complete paths of \( \text{Path}(\text{SemPi}(PM)) \) is called \( \text{PathCO}(PM) \).

In an OBM-\(\pi\) the creation of new variables is done through the restriction operator. The names are new in the scope where they are declared. This allows the same names to appear in a term, but with different scopes. In this way, the same message identifier can appear as labels of different transitions of a path.

All new names of a term must be different to allow its representation as a hypergraph, because each name represents one vertex or one edge. In this way, the paths (that will be considered in comparison between semantics) are paths structurally congruent to complete paths of LTS of the translation of OBHG, where all new names are different. These paths are called \textbf{OB paths}.

**Definition 4.5** Given an OBM-\(\pi\) \(PM\) and a complete path \(c \in \text{PathCO}(PM)\). An \textbf{OB path} equivalent to \(c\) is a path \(b\) where:

i) \( |b| = |c| \);

ii) \( \forall t = (i, l, p), \) where \( l \) is a bound output action, \( p \in \text{TermoOB}_{PM} \);

iii) \( \forall s_j = (i^s_j, l^s_j, p^s_j) \in c, t_j = (i^t_j, l^t_j, p^t_j) \in b, i^t_j \equiv i^s_j \land p^t_j \equiv p^s_j, \) with \( j = 1..|c|\) and \( i^1_t = i^1_s \);

iv) \( \forall s = (i^s, l^s, p^s), t = (i^t, l^t, p^t) \in b, \) where \( l^s = \overline{t}(id^s) \) and \( l^t = \overline{t}(id^t) \) are bound output actions, \( id^s \neq id^t \).

The set of all OB paths of \( \text{Path}(\text{SemPi}(PM)) \) is called \( \text{PathOB}(PM) \).

All complete proofs of theorems in this work may be found in [5].

**Theorem 4.6** Given an OBHG \(HG\) and its translation \(PM = (I_{\pi}, A_{\pi})\), any complete path of \(PM\) can be translated into an equivalent OB path of \(PM\).

**Proof (Sketch)** The initial term and the labels of two paths are the same. The only differences are the bound names in the labels. Due to the rules of structural congruence, the paths that have all subterms structurally equivalent are themselves equivalent. \(\square\)

Given an OBHG and \(PM\) (its translation), there are in \(\text{PathOB}(PM)\) transitions whose labels are \(\tau\) or free output actions, where the finals states are not OB terms. These terms cannot be obtained from translation of some hypergraph. But, the terms resulting from transitions with labels of bound output action are OB terms and the translation of hypergraphs of OBHG LTS results in terms equivalent to them. These terms are only equivalent (and not equal) due to the automatic choice of new names.

To compare the semantics of two models we remove the \(\tau\) transitions and
translate other transitions into transitions of an OBM-π LTS. The TC function translates a path in OBM-π LTS into a path in OBHG LTS removing the τ transitions and translating the valid terms into typed hypergraphs.

**Definition 4.7** Given an OBM-π PM, an OB term \( B_\pi \) of PM and the type hypergraph \( T_\pi \) of PM, where

\[
B_\pi = (\nu o b_1, \ldots, o b_n, m s g_1, \ldots, m s g_m)(\prod_{i=1}^{n} O_{t o_i}(o b_i)|
\prod_{j=1}^{m} @ (t_j, m s g_j, d_j, p_j[1], \ldots, p_j[k]))
\]

the translation from \( B_\pi \) into a hypergraph \( H^T \) is given by function \( \text{Trad}^T_{\pi} : \text{Term}O B_{PM} \to \text{HiperGTip} \) defined by:

\[
\text{Trad}^T_{\pi}(B_\pi) = (H, t^H, T),
\]

where

\[
T = T_\pi, H = (V_H, E_H, sc^H, t^H),
\]

\[
V_H = \{o b_i | 0 < i \leq n\}, E_H = \{m s g_j | 0 < j \leq m\},
\]

\[
sc^H(m s g_j) = \begin{cases} p_j & \text{if } k > 0 \\
\langle \rangle & \text{if } k = 0 \end{cases},
\]

\[
t^H(m s g_j) = d_j, t^H(o b_i) = t_0, \text{ and } t^H(m s g_j) = t_j
\]

**Definition 4.8** Given an OBHG HG and its translation \( PM = \text{Trad}_{HG}(HG) \) and the type hypergraph \( G \) of PM, the path translation is the total function \( TC^G : \text{PathOB}(PM) \to \text{Path}(\text{SemOBHG}(HG)) \) defined by:

\[
TC^G(c) = \begin{cases}
\lambda & \text{if } c = \lambda \lor c = \langle (i, l, p) \rangle \lor \\
\langle (\text{Trad}^G_{\pi}(i), l_1.l_2, \text{Trad}^G_{\pi}(p_2)) \rangle & \text{if } c = \langle (i, l_1, p_1), (p_1, l_2, p_2) \rangle \land \\
TC^G((\langle i, l_2, p_2 \rangle).R) & \text{if } c = \langle (i, \tau, p_1), (p_1, l_2, p_2) \rangle .R \\
TC^G((\langle i, l_1, p_1 \rangle, (p_1, l_2, p_2) \rangle) & \text{if } c = \langle (i, l_1, p_1), (p_1, l_2, p_2) \rangle .R \land \\
TC^G(R) & l_1 \neq \tau \land l_2 \neq \tau
\end{cases}
\]

To prove that the translation \( \text{Trad}_{HG} \) preserves the semantics of OBHG we show that all paths that belong to OBHG LTS also belong to the set of complete paths of its translation and vice-versa.

Each step of a sequential derivation creates several possibilities of derivations (all isomorphic) for the next step, because the result of a pushout is a
set of isomorphic objects. So, for each sequential derivation in $SDer_{HG}$ there exists a set of sequential derivations, isomorphic to it, that belong to $SDer_{HG}$. In theorem 4.9, we prove that for each path in $Path(SemOBHG(HG))$, exists one path in $PathCO(Trad_{HG}(HG))$, that translated is isomorphic to the first.

**Note 6** If $A$ and $B$ are isomorphic objects then we write $A \approx B$.

**Theorem 4.9** Let $HG = (T, I_{HG}, N, n)$ and $Trad_{HG}(HG) = (I_\pi, A_\pi) = PM$. If $c \in Path(SemOBHG(HG))$, then $\exists t \in PathCO(PM)$ such that $TC^T(t) \approx c$.

**Proof (Sketch)** The proof will be by induction on length of path $c$.

If $c = \langle (I, l, G) \rangle$, then there is one derivation from $I$ to $G$, with the application of rule $nr$. The match $m$ is built from $id \in I$. $G$ is the pushout of $r$ and $m$, therefore has all the elements in $I$ (except $id$) and all elements created by $nr$. Since $I$ is translated into $I_\pi$, we know that exists, in $I_\pi$, a message $id$ and an object $v$ that synchronize many times until two visible actions $\overline{m}$ and $\overline{t}(id)$ are performed, resulting in a term $T$ that contains all agents in $I_\pi$ that are not involved in treatment of message $id$, all created agents and the object agent that treat the message. Translating this path we obtain a path equivalent to $c$.

In theorem 4.9 we prove that all paths in OBHG LTS are in set of complete paths of OBHG translated. Now we will prove the inverse. In theorem 4.10, we prove that for all paths in $PathCO(Trad_{HG}(HG))$, exists translation of this path is in $Path(SemOBHG(HG))$.

**Theorem 4.10** Let $HG = (T, I_{HG}, N, n)$, $Trad_{HG}(HG) = (I_\pi, A_\pi) = PM$ and $T$ the type hypergraph of $PM$. If $t \in PathCO(PM)$, then $TC^T(t) \in Path(SemOBHG(HG))$.

**Proof (Sketch)** The proof will be by induction on the number of labels of bound outputs.

If $t = \langle (I_\pi, l_0, T_0), \ldots, (T_n, \overline{m}, T_{n+1}), (T_{n+1}, \overline{t}(id), T_{n+2}) \rangle$, then exists in $I_\pi$ two agents $id$ and $v$ that synchronize creating this path and resulting in $T_{n+2}$. For the translation of $t$ in OBHG LTS to exist, the following diagram must be a derivation:

\[
\begin{array}{c}
\text{Trad}^T_\pi(I_\pi) \xrightarrow{r'} \text{Trad}^T_\pi(T_{n+2}) \\
\downarrow \text{PO} \quad \downarrow m' \\
L^T \xrightarrow{r} R^T \\
\downarrow m \\
\end{array}
\]
This is true because there are in \( T_{n+2} \) all agents in \( I_\pi \) that are not involved in treatment of \( id \), all agents created in this treatment and the object agent that treated \( id \). Since this is the result of the pushout of \( r \) and \( m \) in the typed hypergraph category, then \( \text{Trad}^T_\pi (T_{n+2}) \) is the result of application of \( r \) in \( \text{Trad}^T_\pi (I_\pi) \).

\[ \square \]

5 Final Remarks

In this work we have proposed a translation from object-based hypergraph grammars into \( \pi \)-calculus agents, more specifically, into agents that have a specific form described by the OBM-\( \pi \) model. We compared the LTSs of the two models to prove that the translation preserves the OBHG semantics. However, there is a change of granularity in the translation: an action that was performed atomically in OBHG (one rule application) is performed in many steps when we consider its translation into \( \pi \)-calculus. This means that not all sequences of transitions of a translated OBHG correspond to computations of the original OBHG. Therefore, if we use the translated model to prove properties, we should take care about the results. If a property is satisfied, then it is also satisfied by the original grammar (because all paths of the OBHG LTS are included in the LTS of the translated OBHG). If a property is not satisfied, it does not necessarily means that the property does not hold for the original OBHG because it could be that it fails for one of the paths of the translated OBHG that does not correspond to a path of the original OBHG. But, as the observable labels of the two LTSs are exactly the same (and occur in the same orders), we conjecture that many of the properties in which we are interested in (that properties involving these labels) are also preserved by the translation. This is currently under investigation.

In this work, we did not consider the object’s internal state. This choice was made because we wanted to define a finite model. The inclusion of attributes could lead to infinite models that do not permit the automatic verification through model checking [6]. In [9] we propose an extension of this model to include some attributes, keeping the finiteness of the model.

The amount of synchronizations needed to represent the rule application can be reduced using the polyadic \( \pi \)-calculus [7], where we can pass through a channel several names at the same time. This extension can be easily carried out.

If one wants to use \( \pi \)-calculus tools to verify properties of the systems specified using OBHGs, the user must know the language of specification of properties used by that tool, for example \( \pi \)-logics [3]. The user does not actually need to know how the translation was done, because the labels of the
transition systems, that are the events over which we can define properties, are the same in both transition systems (and the occurrences of these events are in the same orders). However, it could be useful to have a graphical logical language to describe the properties that is closer to the specification language OBHGs.

References


