Static Analysis of Dynamic Communication Systems by Partner Abstraction ⋆

Jörg Bauer[1] and Reinhard Wilhelm[2]

[1] Informatics and Mathematical Modelling; Technical University of Denmark; Kongens Lyngby, Denmark; *joba@imm.dtu.dk*.

[2] Informatik; Univ. des Saarlandes; Saarbrücken, Germany; *wilhelm@cs.uni-sb.de*.

Abstract. Prominent examples of dynamic communication systems include traffic control systems and ad hoc networks. Dynamic communication systems are hard to verify due to inherent unboundedness. Unbounded creation and destruction of objects and a dynamically evolving communication topology are characteristic features.
Partner graph grammars are presented as an adequate specification formalism for dynamic communication systems. They are based on the single pushout approach to algebraic graph transformation and specifically tailored to dynamic communication systems.
We propose a new verification technique based on abstract interpretation of partner graph grammars. It uses a novel two-layered abstraction, *partner abstraction*, that keeps precise information about objects *and* their communication partners. We identify statically checkable cases for which the abstract interpretation is even complete. In particular, applicability of transformation rules is preserved precisely. The analysis has been implemented in the `hiralysis` tool. It is evaluated on a complex case study, car platooning, for which many interesting properties can be proven automatically.

1 Introduction

We propose a new static analysis for systems with an unbounded number of dynamically created, stateful, linked objects with a constantly evolving communication topology: *dynamic communication systems*. Prominent examples of such systems are wireless communication-based traffic control systems and ad-hoc networks, which have to meet safety-critical requirements that are hard to verify due to the dynamics and unboundedness of dynamic communication systems. A rather obvious key observation will facilitate both the specification and verification of dynamic communication system later on:

The Partner Principle. The behavior of an object in a dynamic communication system is determined by its state and by the state of its communication partners.

The partner principle drives both the choice of the specification formalism and the design of the abstraction. Dynamic communication systems are intuitively modeled using graph grammars (or graph transformation systems), because a state of a dynamic communication system can be modeled as a graph, hence the evolution of states as graph grammar rules. Graph grammars come in many flavors, and we refer to [1] for an overview. Our specification formalism, *partner graph grammars*, employs the single pushout approach to graph transformation with a restricted form of negative application conditions, called *partner constraints*, to model dynamic communication systems. Partner constraints restrict the applicability of rules and reflect the partner principle.

⋆ This work was supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS). See `www.avacs.org` for more information.

A partner graph grammar G consists of a set of rules and an initial graph. The *graph semantics* $[\![\mathsf{G}]\!]$ of G is the set of all graphs generated by application of rules starting from the initial graph. For dynamic communication systems, there are typically an infinite number of graphs of unbounded size, making verification a hard task.

We aim at automatically computing a bounded over-approximation $[\![\mathsf{G}]\!]^\alpha$ of $[\![\mathsf{G}]\!]$. Our technique is based on abstract interpretation [2], where two things need to be defined. First, an *abstraction* $\alpha(G)$ for a single graph G; second, *abstract transformers*, *i.e.*, how to apply rules to abstract graphs.

Abstraction of graphs is called *partner abstraction*. As the name suggests, it is again motivated by the partner principle and summarizes objects that are conjectured to behave in the same way, namely, objects in the same state with similar communication partners. As abstract transformers, we define *best abstract transformers* in the spirit of [3]. Though not computable in general, we show it can be done for partner graph grammars using the concept of *materialization*, a restricted form of concretizations.

Contributions. We present an intuitive specification formalism for dynamic communication systems and an implementation of our analysis that allows to verify topology properties of them. Using graph grammars to specify and verify dynamic communication systems is, to the best of our knowledge, novel. Using our tool, we analyzed the complex platoon case study (*c.f.*, Section 1.1) that earlier approaches failed to verify. Moreover, the tool proves to be well-suited for system design.

On the theoretical side, we present a static analysis of partner graph grammars, which can handle negative application conditions. It is based on a novel abstraction called partner abstraction. Our analysis is shown sound and even complete for some well-defined cases that occur in practice. In particular, we obtain a result stating the exact preservation of rule applicability by partner abstraction.

Outline. First, we present our running example: car platooning. After that, we introduce partner graph grammars as an adequate specification formalism for dynamic communication systems. Section 3 describes the abstract interpretation of partner graph grammars by partner abstraction. Section 4 reports on our implementation and experiments with it, before we comment on related work and conclude.

1.1 Case Study: Car Platooning

Our case study is taken from the California PATH project [4], the relevant part of which is concerned with cars driving on a highway. In order to make better use of the given space, cars heading for the same direction are supposed to drive very close to each other building *platoons*. Platoons can perform actions like merging or splitting. There are many features that make verification difficult: destruction and dynamic creation of cars, *i.e.*, driving onto and off the highway, an evolving and unbounded communication topology, or concurrency. All the verification methods developed in [4] are inappropriate, because they consider static scenarios with a fixed number of cars only.

A *platoon* consists of a *leader*, the foremost car, along with a number of *followers*. A leader without any followers is called *free agent* and is considered a special platoon. Within a platoon there are communication channels between the leader and each of its followers. Inter-platoon communication is only between leaders. As a running example, the platoon *merge maneuver* is studied. It allows two approaching platoons to merge. The merge maneuver is initiated by opening a channel between two distinct platoon

leaders, *i.e.* leaders or free agents. Then, the rear leader passes its followers one by one to the front leader. Finally, when there are no followers left to the rear leader, it becomes itself a follower to the front leader.

A Partner Graph Grammar for Platoon Merge. The merge maneuver is straightforward to model as partner graph grammar with nodes representing cars and edges representing communication channels. Internal states of cars are modeled as node labels. The rules $\mathsf{R}_{\mathrm{merge}}$ of a partner graph grammar modeling the merge maneuver are given in Figure 1. We refer to this figure for an intuitive understanding. The formal notions are introduced in Section 2. We employ five node labels in $\mathsf{R}_{\mathrm{merge}}$. Three of the labels – ld, flw, and fa – represent the states of a car being a leader, follower, or free agent, respectively. Two labels – rl and fl – are used to model situations that occur during a merge maneuver. They distinguish the leader of the rear and the front platoon during a merge. Note that the physical position of platoons is not modeled but abstracted by nondeterminism.

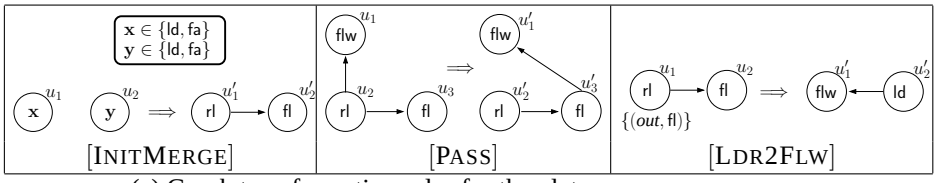
The [INITMERGE] rule models the initiation of a merge maneuver. In fact, the rule stated in Figure 1 is a shorthand denoting four rules, one for each possible combination of leaders and free agents. Followers are handed over from the rear to the front leader in rule [PASS]. Eventually, after passing all followers, the rule [LDR2FLW] can be applied yielding a merged platoon. It makes use of a *partner constraint* requiring the rear leader not to have any followers left. The partner constraint is the set attached to node u_1 in rule [LDR2FLW]. It restricts the application of this rule to cases, where the rear leader has an outgoing edge to a front leader and no other incident edges. There are two simple rules, [CREATE] and [DESTROY], that are not given in Figure 1 but belong to $\mathsf{R}_{\mathrm{merge}}$. The [CREATE] rule has an empty left graph and a single fa-labeled node as right graph. It caters for unconstrained creation of free agent cars. The [DESTROY] rule is the inverse rule, whose application removes a free agent.

Part (b) of Figure 1 shows a sample graph generated by $\mathsf{R}_{\mathrm{merge}}$. Subgraphs **A**, **B**, and **F** are free agent platoons. **C** and **D** are platoons of three and four cars, respectively, whereas **E** depicts a snapshot during a merge maneuver. All these subgraphs are connected components of the graph. As connected components become crucial later on, we shall also call them *clusters*. Cluster $\mathbf{D}$ may evolve from clusters $\mathbf{B}$ and $\mathbf{C}$ by the application of [INITMERGE] and a subsequent application of [LDR2FLW]. For the application of [INITMERGE], $\mathbf{x}$ in this rule is set to fa and $\mathbf{y}$ to ld. This represents the case, where the free agent approaches the platoon from behind.

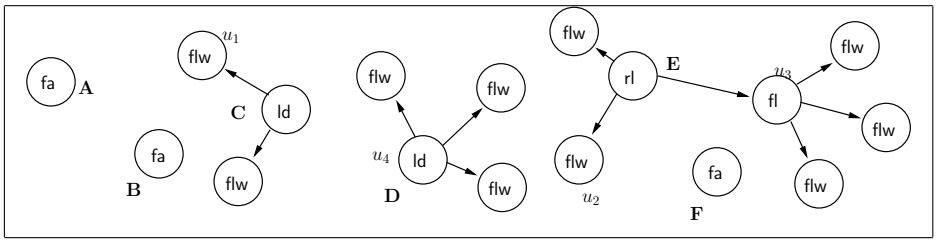
We are interested in proving topology properties of the platoon merge maneuver. Examples of such properties are: (i) cars have a unique leader, unless they are free agents, or (ii) the asymmetry of the leadership relation. The latter means that there are no two cars considering each other to be their leader.

2 Partner Graph Grammars

The purpose of this section is to introduce our formalism for the specification of dynamic communication systems. It is called *partner graph grammars* and is based on the single-pushout approach to graph transformation. Partner graph grammars are restricted to injective matches. They feature a restricted form of negative application conditions called partner constraints intuitively described in Section 1.1. They are an important feature for concise specifications, because they allow to express, when a rule must *not* be applied.



(a) Graph transformation rules for the platoon merge maneuver.



(b) A sample communication topology.

Fig. 1. The platoon merge maneuver. Part (a) shows three graph transformation rules: the initiation of a merge, the hand-over of followers from rear to front leader, and the end of a merge. Part (b) shows a graph generated from an empty graph by these rules. Node labels are shown inside nodes.

2.1 Preliminaries

We recall briefly some standard concepts and notations for reasoning about graphs. Let $\mathcal{L}$ be a finite set of *node labels*. A finite *directed, node-labeled graph*—or graph—G over $\mathcal{L}$ is a triple (V, E, lab), where V is a finite set of nodes, $E \subseteq V \times V$ is a set of edges, and $\mathsf{lab} : V \to \mathcal{L}$ is a labeling mapping. The set of nodes, edges, and the labeling of a graph G are written V_G, E_G, and lab_G, respectively. The set of all finite graphs over $\mathcal{L}$ is written $\mathcal{G}(\mathcal{L})$. The unique graph without nodes is called the *empty graph* and written E. The disjoint graph union of G_1 and G_2 is written $G_1 \dot\cup G_2$, where $\dot\cup$ is also used for disjoint set union. Let $G \in \mathcal{G}(\mathcal{L})$ be a graph and $v \in V_G$ be a node. The set of *incoming partners* of v is defined to be $\triangleright^G v = \{w \in V_G \mid (w, v) \in V_E\}$. Analogously, the set $v\triangleright^G$ of *outgoing partners* and the set of *partners* $\mathsf{pa}_G(v) = \triangleright^G v \cup v\triangleright^G$ are defined. Let $G \in \mathcal{G}(\mathcal{L})$ be a graph and $\mathcal{R} \subseteq V_G \times V_G$ an equivalence relation, such that $v_1 \mathcal{R} v_2$ implies $\mathsf{lab}_G(v_1) = \mathsf{lab}_G(v_2)$. The *quotient graph* $G/\mathcal{R}$ is defined to be the graph H with $V_H = V_G/\mathcal{R}$, $E_H = \{([v_1], [v_2]) \mid (v_1, v_2) \in E_G\}$, and $\mathsf{lab}_H = \lambda[v].\mathsf{lab}_G(v)$. Two nodes $v_1, v_2 \in V_G$ are *connected*, iff there exist $u_1, \ldots, u_n \in V_G$, where $u_1 = v_1$, $u_n = v_2$, and $u_i \in \mathsf{pa}_G(u_{i+1})$ for $1 \leq i < n$. Graph G is *connected*, iff all its nodes are pairwise connected. Let $G, H \in \mathcal{G}(\mathcal{L})$ be graphs. A mapping $h : V_G \to V_H$ is called a *morphism*, iff $\mathsf{lab}_G(v) = \mathsf{lab}_H \circ h(v)$ for all $v \in V_G$, and $(h(v_1), h(v_2)) \in E_H$ for all $(v_1, v_2) \in E_G$. Graph G is a *subgraph* of H, written $G \leq H$, iff there exists an injective morphism from G to H. G and H are isomorphic, written $G \cong H$, if $G \leq H$ and $H \leq G$. A connected graph $G \leq H$ is a *connected component* of H, iff $G' = G$ for all connected graphs G' with $G \leq G' \leq H$. The set of all connected components of H is written $\mathsf{cc}(H)$. Often, we use the term *cluster* instead of connected component.

2.2 Definition

Let $\mathcal{L}$ be an arbitrary finite set of node labels in the remainder. A partner graph grammar is a pair (R, I) of a set of *graph transformation rules* and an *initial graph*. Each transformation rule is a four-tuple (L, h, p, R), where L and R are graphs and h maps nodes in L to nodes in R injectively. The mapping p may associate a *partner constraint* with a node in L. A rule *matches* another graph G, iff L is a subgraph of G due to an injective morphism m—called *match*— and if partner constraints are satisfied. For a given node v in L, the partner constraint $p(v)$ restricts the possible sets of partners of $m(v)$ in G.

If a transformation rule (L, h, p, R) matches graph G due to match m, it can be *applied to* G. The result of the application is the graph H, whose nodes are computed as follows: For each node $v \in V_L$ that is not in the domain of h, the node $m(v)$ is removed from G. On the other hand, each node $v \in V_R$ that is not in the codomain of h is disjointly added to the remaining nodes of G, while for each node v in the domain of h, the node $m(v)$ remains in G. The edges of H are obtained by removing edges from G that are incident to disappearing nodes. Moreover, for all edges (v_1, v_2) in V_L, such that both $m(v_1)$ and $m(v_2)$ remain in V_H, the edge $(m(v_1), m(v_2))$ is removed from G, while all edges in R are added. In Definition 1, $\rightharpoonup$ denotes partial mappings

Definition 1 (Partner Graph Grammar).

1. *A* graph transformation rule r *is a four-tuple* (L, h, p, R)*, where* $L, R \in \mathcal{G}(\mathcal{L})$*,* $p : V_L \rightharpoonup \wp(\{in, out\} \times \mathcal{L})$*, and* $h : V_L \rightharpoonup V_R$ *is injective. A* partner graph grammar G *is a pair* (R, I)*, where* R *is a finite set of graph transformation rules and* $\mathsf{I} \in \mathcal{G}(\mathcal{L})$ *is the* initial graph.
2. *Let* $G \in \mathcal{G}(\mathcal{L})$ *be a graph and* (L, h, p, R) *a graph transformation rule. An injective morphism* m *from* L *to* G *is called a* match*, iff it satisfies partner constraints,* i.e.*, if for all* $v \in dom(p)$*:* $p(v) = \{in\} \times \mathsf{lab}_G({\triangleright^G} m(v)) \cup \{out\} \times \mathsf{lab}_G(m(v){\triangleright^G})$.
3. *If* $r = (L, h, p, R)$ *matches* G *by match* m*, the* result of the application *is the graph* H *obtained as follows:*
   - $V_H = (V_G \setminus m(V_L \setminus dom(h))) \dot\cup (V_R \setminus h(dom(h)))$
   - $E_H = (E_G \cap (V_H \times V_H)) \setminus \{(m(u), m(v)) \mid (u, v) \in E_L\} \cup \{(m'(u), m'(v)) \mid (u, v) \in E_R\}$*, where* $m' : V_R \to V_H$ *such that* $m'(v) = m(h^{-1}(v))$*, if* $v \in h(dom(h))$ *and* $m'(v) = v$ *otherwise.*
   - $\mathsf{lab}_H(v) = \mathsf{lab}_G(v)$*, if* $v \notin m(V_L)$*;* $\mathsf{lab}_H(v) = \mathsf{lab}_R(h(m^{-1}(v)))$*, if* $v \in m(V_L)$*; and* $\mathsf{lab}_H(v) = \mathsf{lab}_R(v)$*, if* $v \notin h(dom(h))$.

   In this case, we write $G \rightsquigarrow_r H$.
4. *Let* $\mathsf{G} = (\mathsf{R}, \mathsf{I})$ *be a partner graph grammar. For two graphs* $G, H \in \mathcal{G}(\mathcal{L})$*, we write* $G \rightsquigarrow_{\mathsf{R}} H$*, iff there exists an* $r \in \mathsf{R}$*, such that* $G \rightsquigarrow_R H$*. The* graph semantics $[\![\mathsf{G}]\!]$ *of* G *is the set* $\{G \in \mathcal{G}(\mathcal{L}) \mid \mathsf{I} \rightsquigarrow_{\mathsf{R}}^* G\}$ *of graphs. A sequence* $G_1 \rightsquigarrow_{r_1} \ldots \rightsquigarrow_{r_{n-1}} G_n$ *is called a* derivation of length n.

Example 1. The partner graph grammar $\mathsf{G}_{\mathrm{merge}} = (\mathsf{R}_{\mathrm{merge}}, \mathsf{E})$, where E is the empty graph and $\mathsf{R}_{\mathrm{merge}}$ was specified in Figure 1(a) (except for the simple rules [CREATE] and [DESTROY]), serves as a running example. An element of the graph semantics $[\![\mathsf{G}_{\mathrm{merge}}]\!]$ is shown in Figure 1(b). There, rule [LDR2FLW] does *not* match cluster $\mathbf{E}$, because the partner constraint is not satisfied: The rl-labeled node has both adjacent flw and fl nodes, whereas the partner constraint requires fl's only. Cluster $\mathbf{D}$ may evolve from $\mathbf{B}$ and $\mathbf{C}$ by an application of [INITMERGE] followed by an application of [LDR2FLW].

Remember the partner principle stated in the introduction. It observes that the behavior of an object is determined by its own state and the state of its communication partners. In terms of partner graph grammars the partner principle is reflected by partner constraints. They allow to define the application conditions of rules in terms of objects and their communication partners. In particular, they can express the *absence* of certain communication partners.

We conclude this section by stating an important property of partner graph grammars with *empty initial graphs*. For every graph G in the graph semantics of a partner graph grammar with an empty initial graph, the disjoint graph union $G \dot\cup G$ is also an element of the graph semantics (up to isomorphism). This property is called *graph multiplicity*, and it is often observed in dynamic communication systems. For instance in the platoon case study, it is justified to assume an empty initial highway.

Lemma 1 (Graph Multiplicity). *Let $\mathsf{G} = (\mathsf{R}, \mathsf{E})$ be a partner graph grammar. For any graph G holds: If $G \in [\![\mathsf{G}]\!]$, then there exists an $H \in [\![\mathsf{G}]\!]$ with $H \cong G \dot\cup G$.*

The proof of the lemma is by induction on the length n of a derivation of G from E. If $n = 1$, then the applied rule must have an empty left graph, because the initial graph is empty. It can thus be applied to G, too, yielding $G \dot\cup G$ (up to isomorphism). Assume $n > 1$ and a derivation of length n: $\mathsf{E} \rightsquigarrow \ldots \rightsquigarrow G' \rightsquigarrow_r G$. By the induction hypothesis, $G' \dot\cup G'$ is in $[\![\mathsf{G}]\!]$. As both occurrences of G are not connected, r can be applied to both occurrences independently yielding $G \dot\cup G$.

3 Partner Abstraction

Our static analysis of dynamic communication systems modeled by partner graph grammars is based on abstract interpretation [2]. Accordingly, we need to define the abstraction of a single graph first. After that, we will say how the application of transformation rules is lifted to abstract graphs. We conclude this section by stating soundness and completeness results.

3.1 Abstract Graphs

Abstract graphs are obtained by *partner abstraction*. As the name suggests, this abstraction reflects the partner principle, where we observe the behavior of an object in a dynamic communication system to be determined by its state and the states of its communication partners. It is thus justified to consider two objects *partner equivalent*, if they are in the same state *and* if they have communication partners in the same states.

Definition 2 (Partner Equivalence). *Let $G \in \mathcal{G}(\mathcal{L})$ be a graph. Nodes $u, v \in V_G$ are* partner equivalent, *written $u \bowtie_G v$, iff $\mathsf{lab}_G(u) = \mathsf{lab}_G(v)$, $\mathsf{lab}_G(\triangleright^G u) = \mathsf{lab}_G(\triangleright^G v)$, and $\mathsf{lab}_G(u\triangleright^G) = \mathsf{lab}_G(v\triangleright^G)$. We denote the equivalence class of u* wrt. *partner equivalence by $u \bowtie_G$.*

Note that partner equivalence does not consider the number of adjacent nodes with a given label. It only takes the existence of adjacent nodes and the direction of connecting edges into account. This is a place, where partner abstraction loses some information. It is obvious from Definition 2 that each equivalence class *wrt.* partner equivalence can be uniquely identified by a label and a set of pairs of direction and label. Following [5], we call such characterizing information a *canonical name*.

Definition 3 (Canonical Names). *The set* $\mathsf{Names}(\mathcal{L}) = \mathcal{L} \times \wp(\{in, out\} \times \mathcal{L})$ *is called the set of* canonical names over $\mathcal{L}$. *The* canonical name *of node* u *of graph* G *is written* $\mathsf{can}_G(u) = (\mathsf{lab}_G(u), \{in\} \times \mathsf{lab}_G(\triangleright^G u) \cup \{out\} \times \mathsf{lab}_G(u\triangleright^G))$.

The abstraction of graphs is a hierarchical process. First, for each connected component, *i.e.*, for each cluster of a graph, nodes are replaced by their canonical names, which effectively computes the quotient graph *wrt.* partner equivalence cluster-wise. Moreover, we distinguish between singleton equivalence classes and non-singleton equivalence classes. The latter will be called *summary nodes* borrowing terminology from [5]. In general, it is possible to count up to some k serving as a parameter of the abstraction as shown in [6]. For the purpose of this work, however, $k = 1$ suffices.

The second abstraction step is motivated by the partner principle, too. The behavior of an object does not depend on objects, with which it does not communicate. Therefore, in the second abstraction step, we summarize clusters that are isomorphic after quotient graph building. Here, we do not keep any information about the number of summarized clusters. Note that this step summarizes clusters instead of nodes thus yielding a hierarchical abstraction. The notion of canonical naming proves useful for this step, because isomorphic clusters become equal when replacing nodes by their canonical names.

Definition 4 (Abstract Clusters and Graphs). *Let* $C \in \mathcal{G}(\mathcal{L})$ *be a connected graph. The* partner abstraction *of* C *is the pair* (H, mult)*, such that* H *is a graph with* $V_H = \{\mathsf{can}_C(u) \mid u \in V_C\}$*,* $E_H = \{(\mathsf{can}_C(u), \mathsf{can}_C(v)) \mid (u, v) \in E_C\}$*, and* $\mathsf{lab}_H = \lambda(\mathsf{a}, P).\mathsf{a}$*. The second component is a mapping* $\mathsf{mult} : V_H \to \{1, \omega\}$*, where* $\mathsf{mult}(u) = 1$*, if* $|\, u \bowtie_C| = 1$*, and* $\mathsf{mult}(u) = \omega$*, otherwise. We write* $\alpha_c(C) = (H, \mathsf{mult})$*. The pair* (H, mult) *is called an* abstract cluster*, and a node* $u \in V_H$ *with* $\mathsf{mult}(u) = \omega$ *is called a* summary node*. A set of abstract clusters is called an* abstract graph*. The* partner abstraction *of graph* $G \in \mathcal{G}(\mathcal{L})$ *is the abstract graph* $\alpha(G) = \{\alpha_c(C) \mid C \in \mathsf{cc}(G)\}$.

Example 2. The abstraction of the graph representing a communication topology in Figure 1(b) is presented in Figure 2. Additionally, we show the clusters that are abstracted to the abstract clusters and examples of canonical names of two nodes in the abstract graph.

Some remarks about canonical names are noteworthy:

- All nodes u in abstract cluster $\hat{C} = (C, \mathsf{mult})$ satisfy $\mathsf{can}_C(u) = u$.
- The number of abstract graphs is bounded in terms of the number l of node labels. The maximal number of nodes in an abstract cluster is $n = l \cdot 2^{2l+1}$, *i.e.* there are at most $c = 2^{2n}$ abstract clusters and 2^c abstract graphs.
- Partner abstraction constitutes a morphism. The abstraction

$$\alpha(G) = \{(C_1, \mathsf{mult}_1), \ldots, (C_n, \mathsf{mult}_n)\}$$

  induces a morphism from G to $C_1 \dot\cup \ldots \dot\cup C_n$, that is a mapping from V_G to $V_{C_1} \dot\cup \ldots \dot\cup V_{C_n}$ In the remainder, we shall call it the *induced morphism* ξ.

3.2 Abstract Transformers

Having defined partner abstraction, the next step is to define abstract transformers, *i.e.*, the application of graph transformation rules to abstract graphs. To do so, we will follow

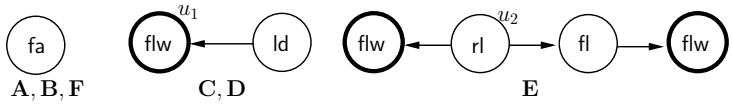


Fig. 2. The partner abstraction of the graph of Figure 1(b). Summary nodes are drawn with a thick rim. The clusters that were summarized to one abstract cluster are given below the respective abstract clusters. Two sample node identities, *i.e.*, canonical names, are stated in the bottom line.

the *best abstract transformer* approach of [3]. Definition 5 first defines the notion of an *abstract match*. It resembles the notion of a match as defined in Definition 1 except for injectivity, which is lost due to summarization of nodes.

If a graph transformation rule r matches an abstract graph $\hat{G}$, we apply it to all concretizations of $\hat{G}$, where a concretization of $\hat{G}$ is a graph G with $\alpha(G) = \hat{G}$. After that, the resulting graphs will be abstracted again using α. In general, there may be infinitely many concretizations of $\hat{G}$. Therefore, we identify a subset of all concretizations guaranteed to be finite for any abstract graph $\hat{G}$. Any such concretization is called a *materialization* as defined in Definition 6. In Lemma 2, we show that materializations define the same abstract transformers as concretizations. Hence, they are a way to compute best abstract transformers.

Definition 5 (Abstract Graph Semantics). *Let $\mathsf{G} = (\mathsf{R}, \mathsf{I})$ be a partner graph grammar, let $r = (L, h, p, R) \in \mathsf{R}$ be a graph transformation rule, and let $\hat{G} = \{(C_1, \mathsf{mult}_1), \ldots, (C_n, \mathsf{mult}_n)\}$ be an abstract graph.*

1. *A morphism m from L to $G := C_1 \dot{\cup} \ldots \dot{\cup} C_n$ is called an* abstract match *from r to $\hat{G}$, iff for all $D \in \mathsf{cc}(L)$ and for all $u \in V_D$ such that $m(u) \in V_{C_i}$ holds: If $u \in \mathit{dom}(p)$, then $p(u) = \{\mathit{in}\} \times \mathsf{lab}_{C_i}(\triangleright^{C_i} m(u)) \cup \{\mathit{out}\} \times \mathsf{lab}_{C_i}(m(u) \triangleright^{C_i})$; and $|m^{-1}(m(u)) \cap V_D| > 1$ implies $\mathsf{mult}_i(m(u)) = \omega$.*
2. *Let $\hat{H}$ be an abstract graph. It is the* result of an application of r to $\hat{G}$, *written $\hat{G} \rightsquigarrow_r^\alpha \hat{H}$, iff there exists an abstract match from L to $\hat{G}$ and there exist graphs M and M', such that $\alpha(M) = \hat{G}$, $M \rightsquigarrow_r M'$, and $\alpha(M') = \hat{H}$.*
3. *The* abstract graph semantics $[\![\mathsf{G}]\!]^\alpha$ *of G is defined inductively as $[\![\mathsf{G}]\!]_0^\alpha = \alpha(\mathsf{I})$, $[\![\mathsf{G}]\!]_{i+1}^\alpha = [\![\mathsf{G}]\!]_i^\alpha \cup \bigcup\{\hat{H} \mid \exists \hat{G} \in [\![\mathsf{G}]\!]_i^\alpha, r \in \mathsf{R}. \hat{G} \rightsquigarrow_r^\alpha \hat{H}\}$, and $[\![\mathsf{G}]\!]^\alpha = \bigcup_{i \geq 0} [\![\mathsf{G}]\!]_i^\alpha$.*

Besides potential non-injectivity of an abstract match, there is an additional requirement regarding summary nodes. If more than one node of the same connected component of the left graph of a rule match the same node in an abstract graph, this node must be a summary node. The application of a matching rule is defined in terms of applying the rule to concretizations. Although this definition is not constructive, we will show how to overcome this by using materializations. The abstract graph semantics collects all abstract clusters obtained by iterated rule applications.

Example 3. The abstract graph semantics $[\![\mathsf{G}_{\mathrm{merge}}]\!]^\alpha$ of the platoon merge implementation is shown in Figure 3. Sample abstract rule applications are

$$\text{(1): } \{\mathbf{C_8}\} \rightsquigarrow^\alpha_{[\textsc{InitMerge}]} \{\mathbf{C_{11}}\} \quad \text{(2): } \{\mathbf{C_9}\} \rightsquigarrow^\alpha_{[\textsc{Pass}]} \{\mathbf{C_{10}}\}$$

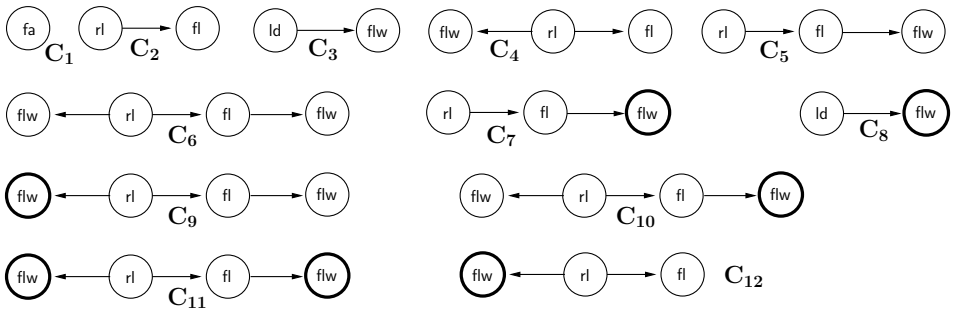


Fig. 3. The abstract graph semantics of the platoon merge partner graph grammar $\mathsf{G}_{\mathrm{merge}}$. It consists of 12 abstract clusters.

In terms of platoons, a concretization justifying (1) contains two platoons of three cars each that merge. They are abstracted to abstract $\{\mathbf{C_8}\}$. The result of the application of $[\textsc{InitMerge}]$ is abstracted to $\{\mathbf{C_{11}}\}$.

We shall now derive upper bounds for the number of concretizations needed to compute abstract transformers. Concretizations within these bounds are called *materializations*. The numbers given in Definition 6 are not always needed, because there are many special cases, where smaller numbers suffice. This is exploited in the implementation of the analysis. In the definition, we use further notations: Given an abstract match m and a node u of abstract cluster (C, mult), $\mathsf{env}(u)$ denotes the number of matched partners of u, *i.e.*, $\mathsf{env}(u) = |\ \mathsf{pa}_C(u) \cap \mathit{codom}(m)\ |$. For an abstraction $\alpha(G) = \hat{G}$ with induced morphism ξ and some node u of $\hat{G}$, we write $\mathsf{mater}(u)$ for the number of nodes mapped to u by ξ, *i.e.*, $\mathsf{mater}(u) = |\ \xi^{-1}(u)\ |$. We call nodes mapped to u by ξ, *nodes materialized from* u. Finally, $\mathsf{matched}(u)$ denotes the number of nodes matching node u of an abstract graph, *i.e.*, for an abstract match m, $\mathsf{matched}(u) = |\ m^{-1}(u)\ |$. Note that the size of all these entities is statically bounded in terms of the shape of left graphs and the number of node labels.

Definition 6 (Materialization). *Let $\hat{G} = \{(C_1, \mathsf{mult}_1), \ldots, (C_n, \mathsf{mult}_n)\}$ be an abstract graph and let $r = (L, h, p, R)$ be a graph transformation rule such that r matches $\hat{G}$ with abstract match m. A concretization G of $\hat{G}$ is called a* materialization *with respect to r and m, iff $|\mathsf{cc}(G)| \leq |\mathsf{cc}(L)| + n$, and for each C_i and all summary nodes $u \in V_{C_i}$: $\max\{2, \mathsf{matched}(u)\} \leq \mathsf{mater}(u)$ and $\mathsf{mater}(u) \leq \mathsf{matched}(u) + 2^{\mathsf{env}(u)+1}$.*

Lemma 2 (Materialization). *If $\hat{G} \rightsquigarrow_r^{\alpha} \hat{H}$ with abstract match m, then there exists a materialization M of $\hat{G}$* wrt. *r and m and graph M' s.t. $M \rightsquigarrow_r M'$ and $\alpha(M') = \hat{H}$.*

The proof of the lemma is based on the observation that nodes in a graph that are not adjacent to a matched node cannot be affected by a rule application. Only matched nodes may change their label and adjacent edges, *i.e.*, only matched nodes *or their partners* may change their canonical name. In any case, at least two or the number of nodes matching a summary node must be materialized from it. If a summary node u is adjacent to $\mathsf{env}(u)$ matched nodes, $2^{\mathsf{env}(u)}$ cases must be distinguished, because a node materialized from u may or may not be connected to each of the matched partners.

Hence, it may or may not be affected by the update. The additional factor of 2 in the upper bound is needed because either one or more than one materialized nodes may be affected in each of the $2^{\mathsf{env}(u)}$ ways. If a summary node u is matched and has matched partners, the number of of nodes matching u must be materialized, too, yielding the additive in the upper bound.

The bound on the number of clusters in a materialization results from the observation that at most $|\mathsf{cc}(L)|$ clusters can be affected by a transformation rule using L. Since there are n abstract clusters, and each needs to be represented in any concretization, the bound for the number of connected components in a materialization is $|\mathsf{cc}(L)| + n$.

Due to the finiteness of all entities bounding the size of materializations also the number of materializations is finite. Lemma 2 also shows that the abstract graph semantics can be computed iteratively in finite time for any partner graph grammar: Since $[\![\cdot]\!]_i^\alpha$ is monotone in i and since there are only finitely many abstract graphs and finitely many materializations of them the computation of the abstract graph semantics will terminate. Besides termination, we obtain the following soundness result.

Theorem 1 (Soundness). *Let* G *be a partner graph grammar. Then the abstract graph semantics is a sound over-approximation of the graph semantics:* $\bigcup_{G \in [\![\mathsf{G}]\!]} \alpha(G) \subseteq [\![\mathsf{G}]\!]^\alpha$.

The proof is obvious, because we defined abstract updates in terms of best abstract transformers. Theorem 1 is typically used in its counterpositive form to prove properties of partner graph grammars, *i.e.*, of dynamic communication systems. If a graph does not occur as a subgraph of an abstract cluster in the abstract graph semantics, it cannot occur in the concrete graph semantics. In our running example, we can thus prove the asymmetry and the uniqueness of the leadership relation.

In fact, Theorem 1 can be used to prove even stronger properties. Assume a transformation rule, where the right graph is a singleton node with an otherwise unused label. If this label does not occur in the abstract graph semantics, it is guaranteed by soundness that this rule never matches in the concrete graph semantics. The additional strength is gained because there may be partner constraints used in the rule.

3.3 Completeness Issues

This section augments the abstract interpretation of partner graph grammars with some unexpected completeness results: First, we identify cases, where partner abstraction preserves precisely the applicability of rules. Second, we present statically checkable criteria that are sufficient for an abstract graph semantics not to contain spurious clusters or even to decide the word problem for a class of partner graph grammars.

It is clear that general completeness results cannot be expected: There is only a bounded number of abstract graphs, whereas the structures in left graphs of transformation rules are unrestricted. However, if we exclude some pathological cases, we can obtain completeness properties.

One pathological case is abstract clusters that include subgraphs like $\mathsf{b} \leftarrow \mathsf{a} \rightarrow \mathsf{b}$, where the a-labeled node is a summary node. Even though we know that all nodes represented by it have an outgoing edge to a b-labeled node, we do not know to which of the two. If such patterns do not occur, we speak of *unique partners*. Furthermore, edges

between summary nodes are a source of information loss. For example, the following cycle between summary nodes may result from abstracting a cycle of any even length of alternating a and b-labeled nodes: $\mathsf{a} \rightleftarrows \mathsf{b}$.

Definition 7 (Special Graphs). *Let* (C, mult) *be an abstract cluster. It has* unique partners*, iff for all* $u \in V_C$ *and for all* $\mathsf{a} \in \mathcal{L}$ *both* $|\triangleright^C u \cap \mathsf{lab}_C^{-1}(\mathsf{a})|$ *and* $|u \triangleright^C \cap \mathsf{lab}_C^{-1}(\mathsf{a})|$ *are at most 1. It has a* summary cycle*, iff there exists an* $n > 1$ *and a subgraph of* C *made up of* n *distinct summary nodes and at least* n *distinct edges among them.*

The definition of summary cycles may seem awkward. It is justified, however, because we are really interested in cycles, where the direction of the edges does not matter (see the proof of Theorem 2 for details).

It is obvious that partner abstraction preserves the applicability of rules, *i.e.*, if a rule matches G, it also matches $\alpha(G)$. Without partner constraints, this holds for any homomorphic abstraction, whereas partner abstraction additionally guarantees the preservation of partner constraint satisfaction. The inverse direction, *i.e.* if a rule matches $\alpha(G)$ it also matches G, does not hold in general. Consider the example of a cycle between two summary nodes above. This abstract cluster is matched by a rule with a left graph being a cycle of length 8 of alternating a and b-labeled nodes. However, this rule does not match the concretization being a cycle of length 6. Theorem 2 describes cases when this direction holds. For simplicity, it is formulated in terms of abstract clusters.

Theorem 2 (Match). *Let* $r = (L, h, p, R)$ *be a transformation rule, where* L *is connected, and let* $\hat{C} = (C, \mathsf{mult})$ *be an abstract cluster. If there is an injective abstract match from* L *to* $\{\hat{C}\}$*, and* $\hat{C}$ *has unique partners and no summary cycles, then* r *matches all* D *with* $\alpha_c(D) = \hat{C}$*.*

The proof exploits two key observations. First, unique partners imply the following. If $(u, v) \in E_C$, and $\mathsf{mult}(u) = 1$ or $\mathsf{mult}(v) = 1$, then for all $u' \in \xi^{-1}(u)$ and all $v' \in \xi^{-1}(v)$, also $(u', v') \in E_D$. W.l.o.g, assume $\mathsf{mult}(v) = 1$. The observation holds, because we know that all u' must have an outgoing edge to a $\mathsf{lab}_C(v)$-labeled node. As there is only one such partner in D, u' must be connected to v'. This results implies that abstract matches that do not contain edges between summary nodes can be mimicked in any concretization. For edges between summary nodes, we use the second key observation: As there are no summary cycles, we can organize matched summary nodes in a forest. The concrete match of L to D is then constructed by traversing this forest. If summary node u is the root of a tree, we choose an arbitrary $u' \in \xi^{-1}(u)$ for the concrete match. If u is not a root, it has only one edge to an ancestor in the traversal. Due to the unique partner property, we can then always pick one u' in D that is connected to the pick we made for the ancestor.

We will now define the notion of *complete clusters* and will show in Theorem 3 that complete clusters imply completeness of the abstract interpretation of partner graph grammars. Since we need to apply the cluster multiplicity property stated in Lemma 1, we concentrate on partner graph grammars with an empty initial graph. We shall call a rule with an empty left graph a *create rule*, because the right graph may be created in an unconstrained way. The most intricate notion we need in Definition 8 is the notion of an *inductive summary node*. A summary node u of abstract cluster $\hat{C}_0 = (C, \mathsf{mult}) \in [\![\mathsf{G}]\!]^\alpha$

is inductive, iff there exists $\hat{C}_n = (C, \mathsf{mult}') \in [\![\mathsf{G}]\!]^\alpha$, where $\mathsf{mult}'(u) = 1$. Moreover, there need to be a set $\hat{C}_i = (C_i, \mathsf{mult}_i)$ of abstract clusters, such that $u \in V_{C_i}$ for all $0 < i < n$, and $\hat{C}_{i+1}$ results from a rule application that does not match u from $\hat{C}_i$. Finally, exactly once, one additional node must become partner equivalent to u by a rule application on this path. Informally, an inductive summary node is part of a loop incrementing its size by 1.

Example 4. The summary node of abstract cluster $\mathbf{C_8}$ of Figure 3 is inductive with clusters $\mathbf{C_3}$ and $\mathbf{C_5}$. In terms of platoons, we obtain arbitrarily many followers by constantly merging with a free agent.

Definition 8 (Complete Clusters). *Let $\mathsf{G} = (\mathsf{R}, \mathsf{E})$ be a partner graph grammar. Abstract cluster $\hat{C} \in [\![\mathsf{G}]\!]^\alpha$ is* complete, *iff*

1. *There exists a create rule (E, h, p, R), such that $\alpha_C(R) = \hat{C}$ and $\hat{C}$ does not contain any summary nodes.*
2. *$\hat{C}$ contains exactly one summary node u, such that u is inductive with complete clusters without summary nodes.*
3. *$\hat{C}$ results from a rule application to complete abstract clusters, such that no summary nodes are matched in the application or created by the application.*

Proving the completeness of the clusters of an abstract graph semantics amounts to imposing a strict order on the clusters, called a *generating order*. Minimal elements are those that result from the application of a create rule.

Example 5. All clusters of $[\![\mathsf{G}_{\mathrm{merge}}]\!]^\alpha$ are complete. Case 1 applies to $\mathbf{C_1}$. Clusters $\mathbf{C_2}$ and $\mathbf{C_3}$ come next in the order and are proven complete by case 3. The same goes for $\mathbf{C_4}$, $\mathbf{C_5}$, and $\mathbf{C_6}$. As mentioned above, case 2 applies to cluster $\mathbf{C_8}$. All remaining clusters result from applying [INITMERGE] to $\mathbf{C_8}$ using case 3.

Theorem 3 (Completeness). *Let $\mathsf{G} = (\mathsf{R}, \mathsf{E})$ be a partner graph grammar. If all $\hat{C} \in [\![\mathsf{G}]\!]^\alpha$ are complete, then $\bigcup_{G \in [\![\mathsf{G}]\!]} \alpha(G) = [\![\mathsf{G}]\!]^\alpha$. If, additionally, R is connected for all $(L, h, p, R) \in \mathsf{R}$, then $G \in [\![\mathsf{G}]\!] \Leftrightarrow \alpha(G) \in [\![\mathsf{G}]\!]^\alpha$ (up to isomorphism).*

The proof of the theorem is only sketched here (see [6] for all the details). It is by well-founded induction on the generating order of the clusters and mimics abstract derivations in the concrete graph semantics. First, we need to show the uniqueness—up to isomorphism and number of nodes materialized from summary nodes—of materializations. Then, Theorem 2 and Lemma 1 guarantee that the loop in the definition of inductive summary nodes can be executed arbitrarily many times, always incrementing the actual size of the summary node by 1. All other summary nodes evolve from inductive summary nodes without change of size as ensured by case 3 of Definition 8. Eventually, Lemma 1 together with connected right graphs guarantees that each abstract cluster individually can have an arbitrary number of concretizations independently of other clusters.

For our running example, Theorem 3 implies that we precisely know all the graphs generated by the platoon merge partner graph grammar, because all right graphs of rules in $\mathsf{R}_{\mathrm{merge}}$ are connected. For this particular grammar, we can thus decide the word problem using Theorem 3.

	#rules	#node labels	#edge labels	#pconstraints	#abstract clusters
merge	8	5	1	1	12
split	4	6	1	4	13
combined	12	6	1	5	169
combined+	12	6	1	9	22
queues	30	18	4	34	159
faulty	32	5	1	1	20

Table 1. Experiments conducted with the `hiralysis` implementation of our analysis.

4 Experimental Evaluation

We have implemented the abstract interpretation of partner graph grammars in the `hiralysis` tool. It takes as input a textual representation of a partner graph grammar and produces as output a graphical representation of the abstract graph semantics. In addition to the material presented here, `hiralysis` supports edge labels.

The `hiralysis` tool has been evaluated on the platoon case study. Some numerical results are shown in Table 1. It shows the size of the input in terms of numbers of node and edge labels, number of transformation rules, and number of partner constraints used in the partner graph grammar. Finally, the size of the abstract graph semantics in terms of numbers of abstract clusters is given.

Among the examples, our running example, **merge**, is the simplest. Its abstract graph semantics consisting of 12 clusters was explicitly given in Figure 3. While that figure was drawn by hand, Figure 4 in Appendix A is generated by `hiralysis`. Grammar **split** implements a split maneuver, while the two versions of **combined** combine merging and splitting. Note that **combine** simply takes the union of **merge** and **split**. The larger number of abstract clusters results from interferences, where platoons try to merge and split simultaneously. This is a source of mistakes not taken into account in the original PATH project [4]. Grammar **combined+** repairs these mistakes by introducing additional partner constraints that restrict the parallelism: another hint to the importance of partner constraints. For the repaired protocol, we were again able to verify properties like unique leaders.

Grammar **queues** extends the original PATH specification by introducing buffered communication. An excerpt of the abstract graph semantics shown in Figure 5 in Appendix B demonstrates that quite complicated graphs may be handled by our technique.

Grammar **faulty** augments the simple merge maneuver with non-deterministically disappearing, unreliable communication links. Such a model is helpful in discovering potential failure situations. More examples are reported on in [6].

In the experiments conducted so far, run time was not found to be a problem. All abstract semantics' of Table 1 were computed in fractions of seconds. Even thousands of clusters could be handled within few seconds. However, we only encountered such numbers for grammars, where we had introduced flaws unintentionally. Therefore, they do not occur among the results given. Interestingly, the `hiralysis` output proved extremely valuable for debugging of such grammars because of its graphical output.

Admittedly, the experiments so far are of modest size, because they are all implemented by hand. We are currently exploring the automated generation of `hiralysis`

input from more fine-grained specifications of dynamic communication systems such as those presented in [7]. The expected partner graph grammars may consist of thousands of rules and will give more hints to the scalability of the technique.

5 Related Work

Partner abstraction was inspired by canonical abstraction of [5], which is particularly well-suited to reason about dynamically allocated data structures. There, reachability is crucial, which cannot be expressed by partner abstraction making it a bad choice for analyzing list-like graphs. However, canonical abstraction is flat, that is, it summarizes only nodes. Partner abstraction is two-layered: it summarizes nodes in a first step and clusters in a second step. Even though partner abstraction may be encoded in canonical abstraction (at a price, as elaborated in [6]), our analysis works fully automatic without intricate instrumentation predicates. Moreover, it is tailored to dynamic communication systems, where graph transformation rules seem easier and more intuitive to write down than the predicate update formulas of [5].

In the very active area of verification of concurrent parameterized systems, many approaches are popping up. Prominent examples include environment abstraction [8] and regular model checking based approaches [9–11] to name only a few. While these techniques are able to deal with infinite-state systems in general, they are typically concerned with unbounded data domains rather than unbounded evolving communication topologies rendering them orthogonal. Even if they are interested in communication topologies, it seems that our approach is the only one able to handle completely arbitrary graphs. Communication topologies are the focus of an abstract interpretation of the π-calculus [12]. However, we believe that π is too fine-grained to model topology evolution hence requiring encoding of features that can be modeled directly using graph grammars.

An approach to formal verification of graph grammars is given in [13]. However, no abstraction is used there, making it inappropriate for the verification of unbounded systems. Another orthogonal approach [14] proved very successful, but is not based on abstract interpretation. Rather it is based on the unfolding semantics of the given graph grammar [15] and approximates behavior by means of Petri nets. Recently, this approach was equipped with counter-example guided abstraction refinement [16].

The only other abstract interpretation based approach was developed independently in [17] and used for software engineering purposes. The underlying abstraction relies on counting incoming and outgoing labeled edges, *i.e.* it is rather edge-centric compared to our node-centric approach. However, our approach provides clear advantages over [17]: we are not restricted to deterministic graphs, we have an implementation, we have a hierarchical abstraction tailored to dynamic communication systems, and we have completeness results. Most importantly, we support negative application conditions in terms of partner constraints. Without this feature graph grammars are hardly usable for dynamic communication systems.

6 Conclusion

We have presented partner graph grammars as an adequate specification formalism for dynamic communication systems, for which we have defined an abstract interpretation based on a novel, hierarchical abstraction called partner abstraction. The analysis was

shown to be sound and, in some well-defined cases, complete. We have reported on the `hiralysis` implementation of the analysis that has been evaluated on the complex platoon case study – automatically revealing flaws and proving so far unproven topology properties.

Currently we are extending our work on unreliable communication links towards probabilistic models, where links do not fail arbitrarily but as determined by a probability distribution. Also we are implementing the automatic generation of `hiralysis` input from more fine-grained models of dynamic communication systems such as those of [7]. Extending the definition of cluster beyond connected components, making partner abstraction parametric, and deepening the hierarchy in the abstraction are ways of extending the applicability of our approach beyond dynamic communication systems.

References

1. Rozenberg, G., ed.: Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations. World Scientific (1997)
2. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixed points. In: Symp. on Princ. of Prog. Lang., New York, NY, ACM Press (1977) 238–252
3. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Symp. on Princ. of Prog. Lang., New York, NY, ACM Press (1979) 269–282
4. Hsu, A., Eskafi, F., Sachs, S., Varaiya, P.: The design of platoon maneuver protocols for IVHS. Technical Report UCB-ITS-PRR-91-6, University of California, Berkley (1991)
5. Sagiv, M., Reps, T., Wilhelm, R.: Parametric shape analysis via 3-valued logic. ACM Transactions on Programming Languages and Systems **24**(3) (2002) 217–298
6. Bauer, J.: Analysis of Communication Topologies by Partner Abstraction. PhD thesis, Universität des Saarlandes, (2006) Available from http://www2.imm.dtu.dk/∼joba/phd.pdf.
7. Bauer, J., Schaefer, I., Toben, T., Westphal, B.: Specification and verification of dynamic communication systems. In: Proc. of the 6th Conference on Application of Concurrency to System Design (ACSD 2006), IEEE Computer Society (2006)
8. Clarke, E.M., Talupur, M., Veith, H.: Environment abstraction for parameterized verification. In Emerson, E.A., Namjoshi, K.S., eds.: VMCAI. Volume 3855 of Lecture Notes in Computer Science., Springer (2006) 126–141
9. Bouajjani, A., Habermehl, P., Vojnar, T.: Abstract regular model checking. In Alur, R., Peled, D., eds.: CAV. Volume 3114 of Lecture Notes in Computer Science., Springer (2004) 372–386
10. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract regular tree model checking of complex dynamic data structures. In Yi, K., ed.: SAS. Volume 4134 of Lecture Notes in Computer Science., Springer (2006) 52–70
11. Bouajjani, A., Jurski, Y., Sighireanu, M.: A generic framework for reasoning about dynamic networks of infinite-state processes. In Grumberg, O., Huth, M., eds.: In 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. (2007)
12. Venet, A.: Automatic determination of communication topologies in mobile systems. In: Static Analysis Symposium. (1998) 152–167
13. Heckel, R.: Compositional verification of reactive systems specified by graph transformation. In: FASE. (1998) 138–153
14. Baldan, P., Corradini, A., König, B.: Verifying finite-state graph grammars: An unfolding-based approach. In Gardner, P., Yoshida, N., eds.: CONCUR. Volume 3170 of Lecture Notes in Computer Science., Springer (2004) 83–98
15. Baldan, P., Corradini, A., Montanari, U.: Unfolding and event structure semantics for graph grammars. In: FoSSaCS. (1999) 73–89
16. König, B., Kozioura, V.: Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In: Proc. of TACAS '06, Springer (2006)
17. Rensink, A., Distefano, D.: Abstract graph transformation. Electr. Notes Theor. Comput. Sci. **157**(1) (2006) 39–59

A Tool Output – $\mathsf{G}_{\mathbf{merge}}$

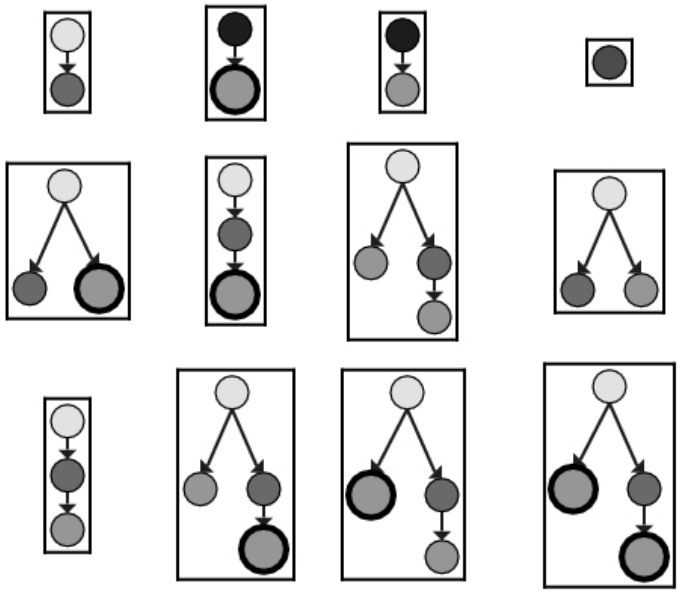


Fig. 4. The abstract graph semantics of the merge implementation $\mathsf{G}_{\mathrm{merge}}$ as generated by the `hiralysis` tools. The hand-drawn version was given in Figure 3.

B Tool Output – Message Buffers Excerpt

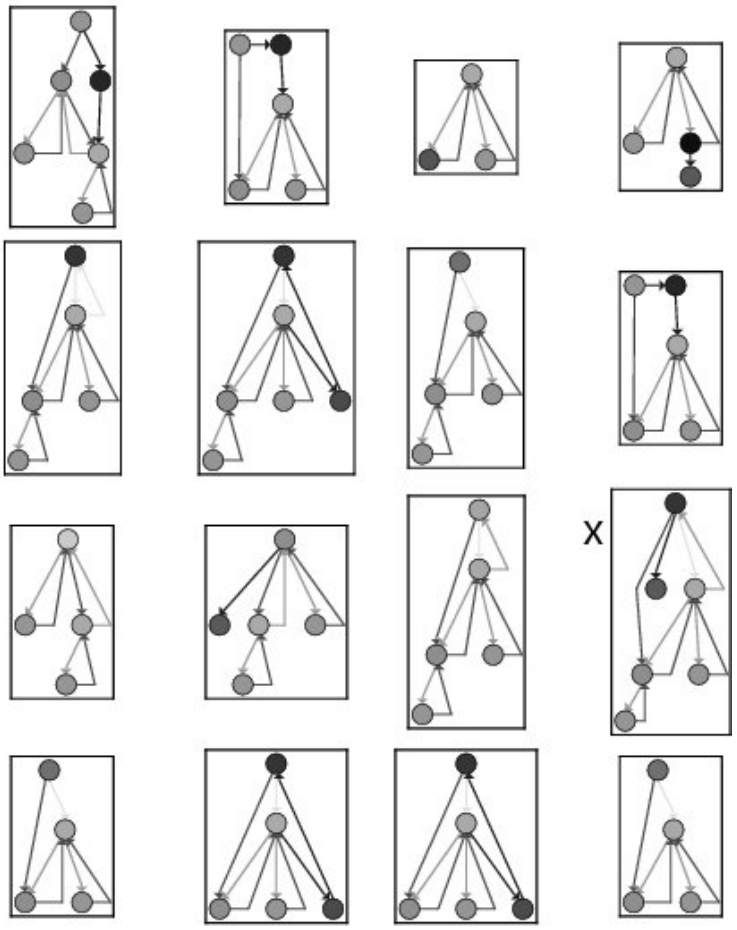


Fig. 5. An excerpt of the 159 abstract clusters constituting the abstract graph semantics of the buffered communication based platoon implementation. This is an example featuring rather involved graphs compared to Figure 4. Consider the X-marked abstract cluster. The four lighter nodes represent two merging platoons. The top dark node is a follower being handed over. Attached to it is a node representing a message pending in the follower's message buffer.