10

Embedding and Local Confluence for Typed AGT Systems

In this chapter, we continue the theory of typed attributed graph transformation systems by describing the Embedding and Extension Theorems, critical pairs and local confluence in Sections 10.1, 10.2, and 10.3, respectively. The constructions have been considered for the graph case in Subsections 3.4.2 and 3.4.3 in Part I.

The notation for the main constructions and results in this chapter is almost identical to that in Chapter 6. However, Chapter 6 is based on an adhesive HLR system AHS, while the present chapter is based on a typed attributed graph transformation system GTS.

10.1 Embedding and Extension Theorems for Typed AGT Systems

In this section, we study the problem of under what conditions a graph transformation $t : G_0 \Rightarrow^* G_n$ can be embedded into a larger context given by a graph morphism $k_0 : G_0 \to G'_0$. In fact, an extension of $t : G_0 \Rightarrow^* G_n$ to a graph transformation $t' : G'_0 \Rightarrow^* G'_n$ is possible only if the extension morphism k_0 is consistent with the given graph transformation $t : G_0 \Rightarrow^* G_n$. This will be shown in the Embedding and Extension Theorems for typed AGT systems below.

This problem has been discussed for the graph case in Part I and presented in the categorical framework in Part II (see Sections 6.1 and 6.2).

First of all we introduce the notion of an extension diagram. An extension diagram describes how a graph transformation $t : G_0 \Rightarrow^* G_n$ can be extended to a transformation $t' : G'_0 \Rightarrow G'_n$ via an extension morphism $k_0 : G_0 \to G'_0$.

Definition 10.1 (extension diagram for typed AGT system). *An* extension diagram *is a diagram (1),*

$$\begin{tikzcd}
G_0 \arrow[r, Rightarrow, "t", "*"'] \arrow[d, "k_0"'] \arrow[dr, phantom, "(1)"] & G_n \arrow[d, "k_n"] \\
G'_0 \arrow[r, Rightarrow, "t'", "*"'] & G'_n
\end{tikzcd}$$

where $k_0 : G_0 \to G'_0$ is a morphism, called an extension morphism, and $t : G_0 \overset{*}{\Rightarrow} G_n$ and $t' : G'_0 \overset{*}{\Rightarrow} G'_n$ are graph transformations via the same productions $(p_0, ..., p_{n-1})$ and via the matches $(m_0, ..., m_{n-1})$ and $(k_0 \circ m_0, ..., k_{n-1} \circ m_{n-1})$, respectively, defined by the following DPO diagrams in $\mathbf{AGraphs_{ATG}}$*:*

$$\begin{tikzcd}
p_i : L_i \arrow[d, "m_i"'] & K_i \arrow[l, "l_i"'] \arrow[r, "r_i"] \arrow[d, "j_i"] & R_i \arrow[d, "n_i"] \\
G_i \arrow[d, "k_i"'] & D_i \arrow[l, "f_i"'] \arrow[r, "g_i"] \arrow[d, "d_i"] & G_{i+1} \arrow[d, "k_{i+1}"] \\
G'_i & D'_i \arrow[l, "f'_i"'] \arrow[r, "g'_i"] & G'_{i+1}
\end{tikzcd} \quad (i = 0, ..., n-1), n > 0$$

For $n = 0$ (see Definition 6.7) the extension diagram is given up to isomorphism by

$$\begin{tikzcd}
G_0 \arrow[d, "k_0"'] & G_0 \arrow[l, "id_{G_0}"'] \arrow[r, "id_{G_0}"] \arrow[d, "k_0"] & G_0 \arrow[d, "k_0"] \\
G'_0 & G'_0 \arrow[l, "id_{G'_0}"'] \arrow[r, "id'_{G_0}"] & G'_0
\end{tikzcd}$$

The following condition for a graph transformation $t : G_0 \overset{*}{\Rightarrow} G_n$ and an extension morphism $k_0 : G_0 \to G'_0$ means intuitively that the boundary graph B of k_0 is preserved by t. In order to formulate this property, we first introduce the notion of a derived span $der(t) = (G_0 \leftarrow D \to G_n)$ of the graph transformation t, which connects the first and the last graph, and later the notion of a boundary graph B for k_0.

Definition 10.2 (derived span). *The* derived span *of an identical graph transformation $t : G \overset{id}{\Rightarrow} G$ is defined by $der(t) = (G \leftarrow G \to G)$ with identical morphisms.*

The derived span *of a direct graph transformation $G \overset{p,m}{\Longrightarrow} H$ is the span $(G \leftarrow D \to H)$ (see Definition 9.4).*

For a graph transformation $t : G_0 \overset{*}{\Rightarrow} G_n \Rightarrow G_{n+1}$, the derived span *is the composition via the pullback (P) in the category* $\mathbf{AGraph_{ATG}}$ *(see below) of the derived spans $der(G_0 \overset{*}{\Rightarrow} G_n) = (G_0 \overset{d_0}{\leftarrow} D' \overset{d_1}{\to} G_n)$ and $der(G_n \Rightarrow G_{n+1}) = (G_n \overset{f_n}{\leftarrow} D_n \overset{g_n}{\to} G_{n+1})$. This construction leads (uniquely up to isomorphism) to the derived span $der(t) = (G_0 \overset{d_0 \circ d_2}{\longleftarrow} D \overset{g_n \circ d_3}{\longrightarrow} G_{n+1})$.*

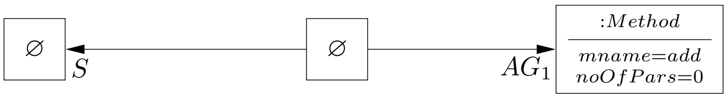
$$\begin{tikzcd}
 & & D \arrow[dl, "d_2"'] \arrow[ddr, "d_3"] & & \\
 & D' \arrow[dl, "d_0"'] \arrow[dr, "d_1"] & (P) & & \\
G_0 & & G_n & D_n \arrow[l, "f_n"'] \arrow[r, "g_n"] & G_{n+1}
\end{tikzcd}$$

In the case $t : G_0 \Rightarrow^* G_n$ with $n = 0$, we have either $G_0 = G_n$ and $t : G_0 \overset{id}{\Rightarrow} G_0$ (see above), or $G_0 \cong G_0'$ with $der(t) = (G_0 \overset{id}{\leftarrow} G_0 \overset{\sim}{\rightarrow} G_0')$.

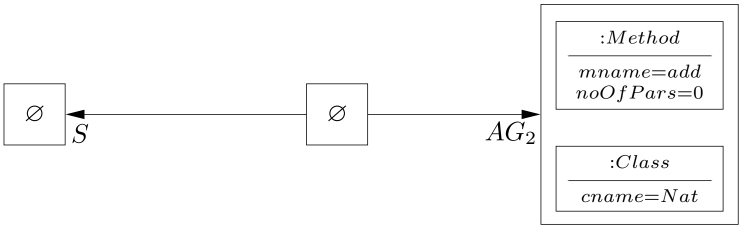
Remark 10.3. According to Section 8.3, we know that pullbacks in $\mathbf{AGraphs_{ATG}}$ exist and that they are constructed componentwise in **Sets**. For the construction in **Sets**, we refer to Fact 2.23. In our construction of derived spans above, the given graph morphisms d_1 and f_n are in $\mathcal{M}$ such that the resulting graph morphisms d_2 and d_3 are also in $\mathcal{M}$. This means that the graph D is the intersection of D' and D_n. Altogether, D can be considered as the largest subgraph of G_0 which is preserved by the graph transformation $t : G_0 \Rightarrow^* G_n$, leading to a derived span $(G_0 \leftarrow D \rightarrow G_n)$ with subgraph embeddings $D \rightarrow G_0$ and $D \rightarrow G_n$.

Example 10.4 (derived span). Here, we construct the derived spans of some of the transformations presented in Example 9.6.

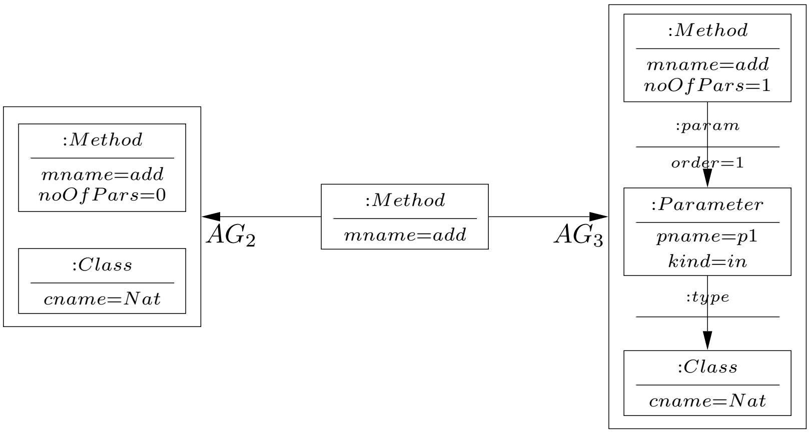
The derived span of the direct transformation $S \overset{addMethod,m_1}{\Longrightarrow} AG_1$ is given by the span of this transformation, as shown below:



For the transformation $S \overset{addMethod,m_1}{\Longrightarrow} AG_1 \overset{addClass,m_2}{\Longrightarrow} AG_2$, the derived span is depicted in the following, where the pullback object is the empty graph:



For the transformation $AG_1 \overset{addClass,m_2}{\Longrightarrow} AG_2 \overset{addParameter,m_3}{\Longrightarrow} AG_3$ with $m_3(n) = 0$, $m_3(p) = p1$, and $m_3(k) = in$, we have the following derived span:



The derived span for the complete transformation $S \overset{*}{\Rightarrow} AG_3$ is the span $\varnothing \leftarrow \varnothing \rightarrow AG_3$. □

In order to define consistency of the extension morphism $k_0 : G_0 \to G_0'$ with respect to the graph transformation $t : G_0 \Rightarrow^* G_n$, we have to define the boundary graph B and, later, also the context graph C for k_0. Intuitively, the boundary B is the minimal interface graph that we need in order to be able to construct a context graph C such that G_0' can be considered as a pushout of G_0 and C via B, written $G_0' = G_0 +_B C$. This construction is given by an "initial pushout over k_0" according to Definition 6.1, and can be constructed explicitly in the category $\mathbf{AGraphs_{ATG}}$ as follows. This construction is given by the initial pushout over k_0 in $\mathbf{AGraphs_{ATG}}$ as defined in Fact 10.7.

The main idea of the construction is similar to that of initial pushouts in the graph case (see Section 6.2).

Definition 10.5 (boundary and context for typed attributed graph morphisms).

1. *Given an attributed graph morphism $f : G \to H$, the boundary–context diagram* (1) *over f in* **AGraphs** *is constructed as follows, where B and C are called the boundary graph and the context graph, respectively, of f, and $b, c \in \mathcal{M}$:*

$$\begin{tikzcd} B \arrow[r, "c\in\mathcal{M}"] \arrow[d, "f"'] \arrow[dr, phantom, "(1)"] & G \arrow[d, "g"] \\ C \arrow[r, "b\in\mathcal{M}"] & H \end{tikzcd}$$

2. *Given a typed attributed graph morphism $f : (G, t^G) \to (H, t^H)$, the boundary–context diagram* (2) *over f in* $\mathbf{AGraphs_{ATG}}$ *is given by the*

boundary–context diagram (1) *over* f *in* **AGraphs**, *where* $t^B = t^G \circ b$ *and* $t^C = t^H \circ c$*:*

$$\begin{tikzcd}
(B, t^B) \arrow[r, "c \in \mathcal{M}"] \arrow[d, "f"'] \arrow[dr, phantom, "(1)"] & (G, t^G) \arrow[d, "g"] \\
(C, t^C) \arrow[r, "b \in \mathcal{M}"] & (H, t^H)
\end{tikzcd}$$

Construction (boundary–context diagram (1)*).* In the following, we denote an attributed graph X by

$$X = (V_G^X, V_D^X, E_G^X, E_{NA}^X, E_{EA}^X, (source_j^X, target_j^X)_{j \in \{G,NA,EA\}}, D^X),$$

where D^X is the $DSIG$-algebra of X, and we denote an attributed graph morphism f by $f = (f_{V_G}, f_{V_D}, f_{E_G}, f_{E_{NA}}, f_{E_{EA}}, f_D)$, where f_D is the $DSIG$-homomorphism of f with $f_{V_D} = \dot\bigcup_{s \in S'_D} f_{D_s}$.

In order to clarify the construction of the boundary graph B, let us recall the signature of an *E-graph* (see Definition 8.1):

$$\begin{tikzcd}
 & E_G \arrow[rr, bend left, "source_G"] \arrow[rr, bend right, "target_G"'] & & V_G & \\
E_{EA} \arrow[ur, "source_{EA}"] \arrow[drr, "target_{EA}"'] & & & & E_{NA} \arrow[ul, "source_{NA}"'] \arrow[dll, "target_{NA}"] \\
 & & V_D & &
\end{tikzcd}$$

The boundary graph B is the intersection of suitable attributed subgraphs B' of G,

$$B = \cap\{B' \subseteq G | D^G = D^{B'}, V_D^G = V_D^{B'}, V_G^* \subseteq V_G^{B'}, E_G^* \subseteq E_G^{B'},$$
$$E_{NA}^* \subseteq E_{NA}^{B'}, E_{EA}^* \subseteq E_{EA}^{B'}\},$$

where the sets V_G^*, E_G^*, E_{NA}^*, and E_{EA}^* built up by the dangling and identification points (see Definition 9.8) are defined as follows:

- $V_G^* = \{a \in V_G^G \mid \exists a' \in E_G^H \setminus f_{E_G}(E_G^G)$ with $f_{E_G}(a) = source_G^H(a')$ or $f_{E_G}(a) = target_G^H(a')\}$
  $\cup \{a \in V_G^G \mid \exists a' \in E_{NA}^H \setminus f_{E_{NA}}(E_{NA}^G)$ with $f_{E_{NA}}(a) = source_{NA}^H(a')\}$
  $\cup \{a \in V_G^G \mid \exists a' \in V_G^G$ with $a \neq a'$ and $f_{V_G}(a) = f_{V_G}(a')\}$;
- $E_G^* = \{a \in E_G^G \mid \exists a' \in E_{EA}^H \setminus f_{E_{EA}}(E_{EA}^G)$ with $f_{E_{EA}}(a) = source_{EA}^H(a')\}$
  $\cup \{a \in E_G^G \mid \exists a' \in E_G^G$ with $a \neq a'$ and $f_{E_G}(a) = f_{E_G}(a')\}$;
- $E_{NA}^* = \{a \in E_{NA}^G \mid \exists a' \in E_{NA}^G$ with $a \neq a'$ and $f_{E_{NA}}(a) = f_{E_{NA}}(a')\}$;
- $E_{EA}^* = \{a \in E_{EA}^G \mid \exists a' \in E_{EA}^G$ with $a \neq a'$ and $f_{E_{EA}}(a) = f_{E_{EA}}(a')\}$.

The context graph C is the attributed subgraph of H defined by

- $V_G^C = (V_G^H \setminus f_{V_G}(V_G^G)) \cup f_{V_G}(V_G^B)$;
- $V_D^C = V_D^H$;
- $E_j^C = (E_j^H \setminus f_{E_j}(E_j^G)) \cup f_{E_j}(E_j^B), \quad j \in \{G, NA, EA\}$;
- $D^C = D^H$.

The attributed graph morphisms $b, c \in \mathcal{M}$, and g are given by

- $b : B \to G$, inclusion with $b_{V_D} = id$ and $b_D = id$;
- $c : C \to H$, inclusion with $c_{V_D} = id$ and $c_D = id$;
- $g : B \to C$, by $g_j(x) = f_j \circ b_j(x), \quad j \in \{V_G, V_D, E_G, E_{NA}, E_{EA}, D\}$.

□

Remark 10.6. Note that $B^* = (V_G^*, V_D^G, E_G^*, E_{NA}^*, E_{EA}^*, (s_j^*, t_j^*)_{j\in\{G,NA,EA\}}, D^G)$ with restrictions $s_j^*(t_j^*)$ of $s_j^G(t_j^G)$, where s and t are abbreviations for *source* and *target*, respectively, is in general not an attributed subgraph of G, such that the subgraph B has to be constructed as the intersection of all subgraphs $B' \subseteq G$ defined above. However, we have the following for B:

- $V_G^* \subseteq V_G^B \subseteq V_G^G$,
- $V_D^B = V_D^G$,
- $E_G^* \subseteq E_G^B \subseteq E_G^G$,
- $E_{NA}^* \subseteq E_{NA}^B \subseteq E_{NA}^G$,
- $E_{EA}^* \subseteq E_{EA}^B \subseteq E_{EA}^G$,
- $D^B = D^G$.

In fact, V_G^B and E_G^B are target domains of operations in B which are proper extensions of V_G^* and E_G^*, respectively, if $s_G^G(E_G^*) \not\subseteq V_G^*$, $t_G^G(E_G^*) \not\subseteq V_G^*$, $s_{NA}^G(E_{NA}^*) \not\subseteq V_G^*$ and $s_{EA}^G(E_{EA}^*) \not\subseteq E_G^*$, respectively. Note that $t_{NA}^G(E_{NA}^*) \subseteq V_D^B$ and $t_{EA}^G(E_{EA}^*) \subseteq V_D^B$ because $V_D^B = V_D^G$.

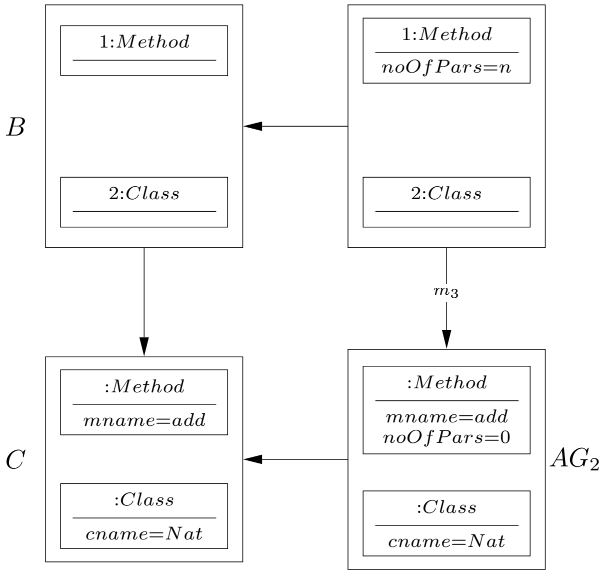
The above constructions are all well defined, leading to an initial pushout over f in $\mathbf{AGraphs_{ATG}}$ (see Definition 6.1).

Fact 10.7 (initial pushouts in AGraphs and $\mathbf{AGraphs_{ATG}}$).

1. *Given an attributed graph morphism $f : G \to H$, the boundary–context diagram* (1) *over f in Definition 10.5 is well defined and is an initial pushout over f in* $(\mathbf{AGraphs}, \mathcal{M})$.
2. *Given a typed attributed graph morphism $f : (G, t^G) \to (H, t^H)$, the boundary–context diagram* (2) *over f is well-defined and is an initial pushout over f in* $(\mathbf{AGraphs_{ATG}}, \mathcal{M})$.

Proof. The construction of the boundary–context diagrams in **AGraphs** and $\mathbf{AGraphs_{ATG}}$ is a special case of the initial-pushout construction in the category **AGSIG-Alg** (see Lemma 11.17) using the isomorphism **AGSIG(ATG)-Alg** $\cong$ $\mathbf{AGraphs_{ATG}}$ of categories, which will be shown in Chapter 11. Note that an explicit proof of initial pushouts for $(\mathbf{AGraphs_{ATG}}, \mathcal{M})$ is more difficult than the general proof for (**AGSIG-Alg**, $\mathcal{M}$). □

Example 10.8 (initial pushout). In the following diagram, the boundary–context diagram of the match morphism m_3 in Example 10.4 is depicted. We have no identification points, but the boundary object B consists of the two dangling points – the *Method* and the *Class* node – because in both cases a node attribute edge is added in AG_2.



□

Now we are able to define the consistency of a morphism k_0 with respect to a graph transformation t. The main idea is that the boundary B of k_0, defined by the initial pushout over k_0, is preserved by the transformation t. This means that there is a suitable morphism $b : B \to D$, where D, defined by the derived span of t, is the largest subgraph of G_0 which is preserved by $t : G_0 \Rightarrow^* G_n$.

Definition 10.9 (consistency). *Given a graph transformation $t : G_0 \overset{*}{\Rightarrow} G_n$ with a derived span $der(t) = (G_0 \overset{d_0}{\leftarrow} D \overset{d_n}{\rightarrow} G_n)$, a morphism $k_0 : G_0 \to G_0'$ in* **AGraphs$_{\mathbf{ATG}}$** *with the initial PO (1) over k_0 is called* consistent *with respect to t if there exists a morphism $b \in \mathcal{M}$ with $d_0 \circ b = b_0$:*

$$\begin{tikzcd}
B \arrow[r, "b_0"] \arrow[rr, bend left, "b"] \arrow[d] \arrow[dr, phantom, "(1)"] & G_0 \arrow[d, "k_0"] & D \arrow[l, "d_0"'] \arrow[r, "d_n"] & G_n \\
C \arrow[r] & G_0' & &
\end{tikzcd}$$

With the following Embedding and Extension Theorems, we can show that consistency is both necessary and sufficient for the construction of extension diagrams. These theorems correspond exactly to the corresponding Theorems 3.28 and 3.29 for the graph case and to Theorems 6.14 and 6.16 for the general case.

Theorem 10.10 (Embedding Theorem for typed AGT systems). *Given a graph transformation $t : G_0 \overset{*}{\Rightarrow} G_n$ and a morphism $k_0 : G_0 \to G'_0$ which is consistent with respect to t, then there is an extension diagram over t and k_0.*

Proof. See Section 11.3. □

Theorem 10.11 (Extension Theorem for typed AGT systems). *Given a graph transformation $t : G_0 \overset{*}{\Rightarrow} G_n$ with a derived span $der(t) = (G_0 \overset{d_0}{\leftarrow} D_n \overset{d_n}{\rightarrow} G_n)$ and an extension diagram (1),*

$$\begin{tikzcd}
B \arrow[r, "b_0"] \arrow[d] \arrow[dr, phantom, "(2)"] & G_0 \arrow[r, Rightarrow, "t", "*"'] \arrow[d, "k_0"] \arrow[dr, phantom, "(1)"] & G_n \arrow[d, "k_n"] \\
C \arrow[r] & G'_0 \arrow[r, Rightarrow, "t'", "*"'] & G'_n
\end{tikzcd}$$

with an initial pushout (2) over $k_0 \in \mathcal{M}'$ for some class $\mathcal{M}'$ closed under pushouts and pullbacks along $\mathcal{M}$-morphisms and with initial pushouts over $\mathcal{M}'$-morphisms, then we have:

1. *k_0 is consistent with respect to $t : G_0 \overset{*}{\Rightarrow} G_n$ with the morphism $b : B \to D_n$.*
2. *There is a graph transformation $G'_0 \Rightarrow G'_n$ via $der(t)$ and k_0 given by the pushouts (3) and (4) below with $h, k_n \in \mathcal{M}'$.*
3. *There are initial pushouts (5) and (6) over $h \in \mathcal{M}'$ and $k_n \in \mathcal{M}'$, respectively, with the same boundary–context morphism $B \to C$:*

$$\begin{tikzcd}
G_0 \arrow[d, "k_0"] \arrow[dr, phantom, "(3)"] & D_n \arrow[l, "d_0"'] \arrow[r, "d_n"] \arrow[d, "h"] \arrow[dr, phantom, "(4)"] & G_n \arrow[d, "k_n"] \\
G'_0 & D'_n \arrow[l] \arrow[r] & G'_n
\end{tikzcd}
\quad
\begin{tikzcd}
B \arrow[r, "b"] \arrow[d] \arrow[dr, phantom, "(5)"] & D_n \arrow[d, "h"] \\
C \arrow[r] & D'_n
\end{tikzcd}
\quad
\begin{tikzcd}
B \arrow[r, "d_n \circ b"] \arrow[d] \arrow[dr, phantom, "(6)"] & G_n \arrow[d, "k_n"] \\
C \arrow[r] & G'_n
\end{tikzcd}$$

Proof. See Section 11.3. □

Example 10.12 (Embedding and Extension Theorems). If we embed the start graph S from Example 10.4 via a morphism k_0 into a larger context H, k_0 is consistent with respect to the transformation $t : S \overset{*}{\Rightarrow} AG_3$. This is due to the fact that S is the empty graph, and therefore the boundary graph is also empty and no items have to be preserved by the transformation. Applying Theorem 10.10 allows us to embed the transformation $S \overset{*}{\Rightarrow} AG_3$, leading to an extension diagram over t and k_0. From Theorem 10.11, we conclude that there is a direct transformation $H \Rightarrow H'$ via the derived span $der(t)$ shown in Example 10.4. The resulting graph H' is the disjoint union of H and AG_3. □

10.2 Critical Pairs for Typed AGT Systems

In order to study local confluence in Section 10.3, we now introduce critical pairs, as discussed in Section 3.4 and used for adhesive HLR systems in Chapter 6.

For the definition of critical pairs, we need the concept of an $\mathcal{E}'$–$\mathcal{M}'$ pair factorization introduced in Section 9.3.

Definition 10.13 (critical pair). *Given an $\mathcal{E}'$–$\mathcal{M}'$ pair factorization in* $\mathbf{AGraphs_{ATG}}$*, a critical pair is a pair of parallel dependent direct transformations $P_1 \overset{p_1,o_1}{\Longleftarrow} K \overset{p_2,o_2}{\Longrightarrow} P_2$ such that $(o_1,o_2) \in \mathcal{E}'$ for the corresponding matches o_1 and o_2.*

In analogy to the graph case considered in Lemma 3.33 and the general case considered in Lemma 6.22, we now show the completeness of critical pairs in $\mathbf{AGraphs_{ATG}}$.

Lemma 10.14 (completeness of critical pairs in $\mathbf{AGraphs_{ATG}}$). *Given an $\mathcal{E}'$–$\mathcal{M}'$ pair factorization where the $\mathcal{M}$–$\mathcal{M}'$ pushout–pullback decomposition property holds (see Definition 5.27), then the critical pairs in* $\mathbf{AGraphs_{ATG}}$ *are complete. This means that for each pair of parallel dependent direct transformations $H_1 \overset{p_1,m_1}{\Longleftarrow} G \overset{p_2,m_2}{\Longrightarrow} H_2$, there is a critical pair $P_1 \overset{p_1,o_1}{\Longleftarrow} K \overset{p_2,o_2}{\Longrightarrow} P_2$ with extension diagrams (1) and (2) and $m \in \mathcal{M}'$:*

$$\begin{tikzcd}
P_1 \arrow[d] & K \arrow[l, Leftarrow] \arrow[r, Rightarrow] \arrow[d, "m"] \arrow[dl, phantom, "(1)"] \arrow[dr, phantom, "(2)"] & P_2 \arrow[d] \\
H_1 & G \arrow[l, Leftarrow] \arrow[r, Rightarrow] & H_2
\end{tikzcd}$$

Remark 10.15. The requirements above are valid in particular for the $\mathcal{E}'_1$–$\mathcal{M}'_1$ and $\mathcal{E}'_2$–$\mathcal{M}'_2$ pair factorizations given in Definition 9.20.

Example 10.16 (critical pairs in *MethodModeling*). In the following, we analyze the critical pairs in our graph grammar *MethodModeling* from Example 9.6. We use the $\mathcal{E}'_2$–$\mathcal{M}'_2$ pair factorization given in Definition 9.20.

For the underlying category $\mathbf{AGraphs_{ATG}}$ with the given type graph *ATG* (see Example 8.9), there is a large number of critical pairs. We have counted 88 different possibilities for only the application of the production *checkNewParameter* in two different ways that lead to a critical pair.

If we analyze all these pairs, we see that most of them are strange in some way and do not meet our intentions for the graph grammar *MethodModeling*. We have aimed at modeling the signatures of method declarations, and our productions reflect this. However, in the critical pairs, often graph nodes have multiple occurrences of the same attribute, or parameters have multiple types or belong to more than one method. All these things are allowed in our general theory of typed attributed graphs, but they do not lead to a consistent method declaration, and cause a high number of critical pairs.

Therefore we analyze only those critical pairs which are of interest for the language of *MethodModeling*, which means that we consider only those graphs that can be derived from the empty graph by applying our productions.

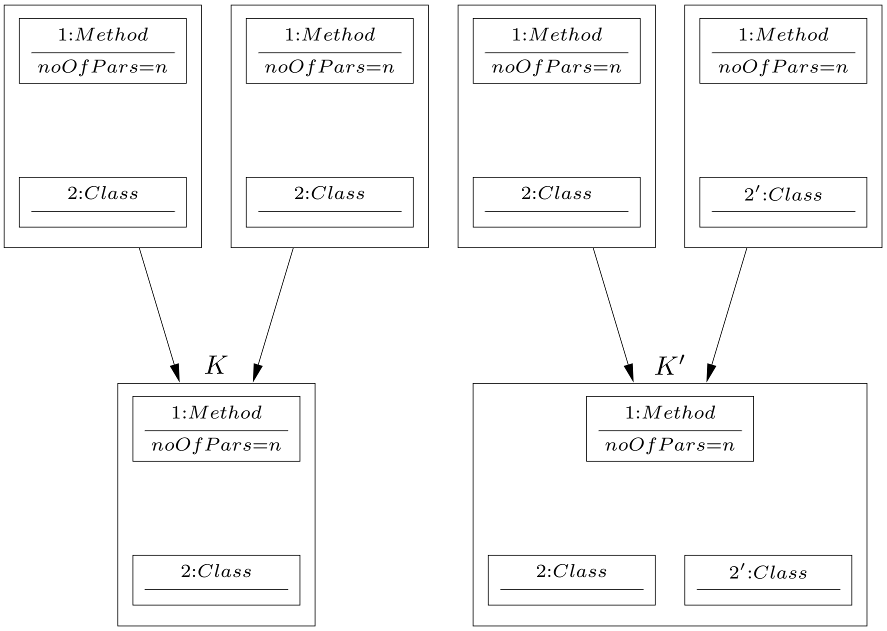
With this restriction, we obtain the following critical pairs $P_1 \overset{p_1,m_1}{\Longleftarrow} K \overset{p_2,m_2}{\Longrightarrow} P_2$:

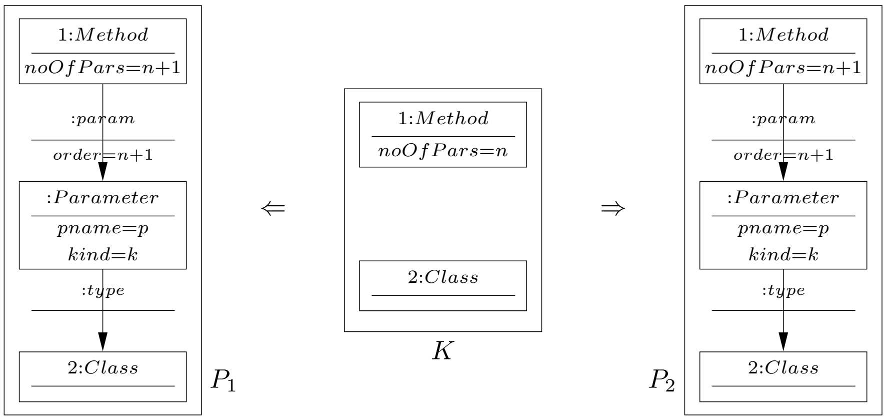
1. $p_1 = p_2 =$ *addParameter*: 2 critical pairs. In this case, two parameters are added to the same method, which increases the number of parameters in the method, i.e. changes the attribute *noOfPars* and causes a conflict. There are two possible cases: the classes of the two new parameters are the same or different.
2. $p_1 =$ *addParameter*, $p_2 =$ *checkNewParameter*: 2 critical pairs. One parameter is added to a method, and another one is deleted. Adding a parameter increases the value of *noOfPars* and the deletion decreases this value, which leads to a conflict. Again there are two possible cases: the classes of the new and deleted parameters are the same or different.
3. $p_1 = p_2 =$ *checkNewParameter*: 2 critical pairs. We delete the last parameter with both transformations. There are then two cases: the parameters that we use for the comparison (which means checking that the last parameter is already in the list) are the same or different.
4. $p_1 =$ *checkNewParameter*, $p_2 =$ *exchangeParameter*: 5 critical pairs. The last parameter of a method is deleted by one transformation, but the other transformation exchanges it with another parameter. There are several different ways in which the parameters involed can be matched.

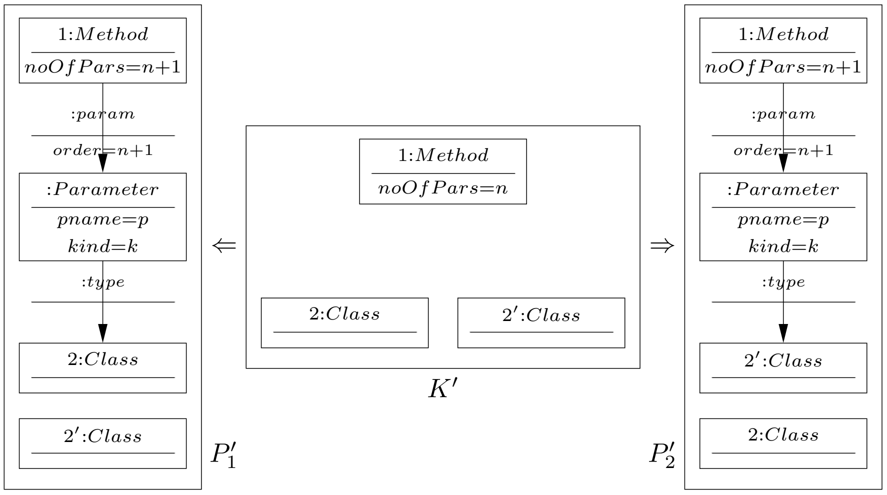
   If one of the exchanged parameters is deleted, the other one can be the same as or different from the parameter we use for the comparison in *checkNewParameter*. This leads to four critical pairs. The fifth critical pair is obtained if both exchanged parameters are mapped together with the deleted one. This is possible, since matches do not have to be injective.
5. $p_1 = p_2 =$ *exchangeParameter*: 11 critical pairs. If the same parameter is exchanged by both transformations, this leads to a conflict. There are 11 cases of how to map at least one parameter of one left-hand side to one parameter of the other left-hand side (including the cases where the matches are not injective).

Other combinations of productions do not result in critical pairs with overlapping graphs being generated by our productions; therefore, altogether, there are 22 critical pairs.

In the following diagram, the two critical pairs $P_1 \Leftarrow K \Rightarrow P_2$ and $P_1' \Leftarrow K' \Rightarrow P_2'$ for the case $p_1 = p_2 =$ *addParameter* are represented. In the top part of the diagram, we show only the left-hand sides of the productions, and the graphs K and K'. In the bottom part, the resulting direct transformations are depicted. The matches are shown by the names of the nodes. The value of the attribute *noOfPars* is changed, which means that in the gluing object (not shown explicitly) of the direct transformations $P_1 \Leftarrow K \Rightarrow P_2$ and $P_1' \Leftarrow K' \Rightarrow P_2'$, there is no node attribute edge between the method and the variable n. Therefore these transformations are parallel dependent. The matches are jointly surjective on the graph part, and both transformation pairs are critical pairs.







□

10.3 Local Confluence Theorem for Typed AGT Systems

A typed AGT system is confluent if, for all pairs of graph transformations starting from the same graph, there are graph transformations that bring the resulting graphs back together. Confluence based on critical pairs has been studied for hypergraphs in [Plu93] and for typed node attributed graphs in [HKT02]. The main result is, that strict confluence of all critical pairs implies local confluence of the whole system. These concepts have been discussed already in Section 3.4 and used for adhesive HLR systems in Chapter 6. Now we instantiate them to typed AGT systems.

With a suitable $\mathcal{E}'$–$\mathcal{M}'$ pair factorization, such as one of those given in Definition 9.20, a graph transformation system is locally confluent if all its critical pairs are strictly confluent.

In Section 3.4, we have shown that local confluence together with termination implies the confluence of the whole system. The termination of typed attributed graph transformation systems will be studied in Section 12.3.

In analogy to the graph case considered in Theorem 3.34 and the general case considered in Theorem 6.28, we are now able to formulate the Local Confluence Theorem for typed AGT systems, based on the concept of strict confluence of critical pairs.

Definition 10.17 (strict confluence of critical pairs). *A critical pair* $K \overset{p_1,o_1}{\Longrightarrow} P_1$*,* $K \overset{p_2,o_2}{\Longrightarrow} P_2$ *is called* strictly confluent *if we have the following conditions:*

1. Confluence. *the critical pair is confluent, i.e. there are transformations* $P_1 \overset{*}{\Rightarrow} K'$, $P_2 \overset{*}{\Rightarrow} K'$ *with derived spans* $der(P_i \overset{*}{\Rightarrow} K') = (P_i \overset{v_{i+2}}{\leftarrow} N_{i+2} \overset{w_{i+2}}{\rightarrow} K')$ *for* $i = 1, 2$.
2. Strictness. *Let* $der(K \overset{p_i,o_i}{\Longrightarrow} P_i) = (K \overset{v_i}{\leftarrow} N_i \overset{w_i}{\rightarrow} P_i)$ *for* $i = 1, 2$, *and let* N *be the pullback object of the pullback (1). There are then morphisms* z_3 *and* z_4 *such that (2), (3), and (4) commute:*

$$\begin{tikzcd}
 & K \arrow[dl, Rightarrow, "{(p_1,o_1)}"'] \arrow[dr, Rightarrow, "{(p_2,o_2)}"] & \\
P_1 \arrow[dr, Rightarrow, "*"'] & & P_2 \arrow[dl, Rightarrow, "*"] \\
 & K' &
\end{tikzcd}$$

$$\begin{tikzcd}
 & & K & & \\
 & N_1 \arrow[ur, "v_1"] \arrow[dl, "w_1"'] & (1) & N_2 \arrow[ul, "v_2"'] \arrow[dr, "w_2"] & \\
P_1 & (2) & N \arrow[ul, "z_1"'] \arrow[ur, "z_2"] \arrow[dl, "z_3"] \arrow[dr, "z_4"'] & (3) & P_2 \\
 & N_3 \arrow[ul, "v_3"] \arrow[dr, "w_3"'] & (4) & N_4 \arrow[ur, "v_4"'] \arrow[dl, "w_4"] & \\
 & & K' & &
\end{tikzcd}$$

Theorem 10.18 (Local Confluence Theorem for typed AGT systems). *Given a graph transformation system GTS based on* ($\mathbf{AGraphs_{ATG}}$, $\mathcal{M}$) *with an* $\mathcal{E}'$–$\mathcal{M}'$ *pair factorization such that* $\mathcal{M}'$ *is closed under pushouts and pullbacks along* $\mathcal{M}$*-morphisms and the* $\mathcal{M}$–$\mathcal{M}'$ *pushout–pullback decomposition property holds, then GTS is locally confluent if all its critical pairs are strictly confluent.*

Proof. See Theorem 11.14 in Section 11.3. □

Remark 10.19. The language L of a graph transformation system GTS is locally confluent if we consider only those critical pairs $P_1 \Leftarrow K \Rightarrow P_2$, where K is an $\mathcal{M}'$-subgraph of a graph G which can be derived from the start graph S. "K is $\mathcal{M}'$-subgraph of G" means that there is an $\mathcal{M}'$-morphism $m : K \to G$.

In Section 11.3, we show that the Local Confluence Theorem is valid for a typed AGT system over the category **AGSIG-Alg** with a well-structured $AGSIG$ and a suitable $\mathcal{E}'$–$\mathcal{M}'$ pair factorization. This implies Theorem 10.18. The requirements for $\mathcal{M}'$ are valid for $\mathcal{M}' = \mathcal{M}'_1$ and $\mathcal{M}' = \mathcal{M}'_2$ as considered in Example 9.22.

Example 10.20 (local confluence in $MethodModeling$**).** The task of analyzing the local confluence of our graph grammar $MethodModeling$ is extensive owing to the fact that there are so many critical pairs, even if we consider only the language of *MethodModeling* (see Example 10.16). Therefore we only give arguments for local confluence here, and do not prove it.

The confluence of the critical pairs can be shown relatively easily. In the following we describe how to find suitable productions and matches for a critical pair $P_1 \overset{p_1,m_1}{\Longleftarrow} K \overset{p_2,m_2}{\Longrightarrow} P_2$ that lead to confluence.

1. $p_1 = p_2 = addParameter$. In this case two parameters are added to the same method, which increases the number of parameters in the method. We can apply p_2 to P_1 with a match m_2' slightly different from m_2, where only the value of the attribute $numberOfPars$ is increased by 1. This works similarly if p_1 is applied to P_2 with a match m_1'. We then obtain transformations $P_1 \overset{p_2,m_2'}{\Longrightarrow} X$ and $P_2 \overset{p_1,m_1'}{\Longrightarrow} X$.
2. $p_1 = addParameter$, $p_2 = checkNewParameter$. Here a parameter is added and deleted in the same method. In P_1, we can use the production *exchangeParameter* to swap the last two parameters and then apply p_2 to this graph with a match m_2' similar to m_2, where only the value of the attribute $numberOfPars$ in this method is increased by 1. Applying p_1 to P_2 results in a common graph. This means that we have transformations $P_1 \overset{exchangeParameter}{\Longrightarrow} X' \overset{p_2,m_2'}{\Longrightarrow} X$ and $P_2 \overset{p_1,m_1'}{\Longrightarrow} X$.
3. $p_1 = p_2 = checkNewParameter$. We have deleted the same (last) parameter with both transformations, and therefore it already holds that $P_1 = P_2$ or $P_1 \cong P_2$.
4. $p_1 = checkNewParameter$, $p_2 = exchangeParameter$. In K, a parameter is deleted by p_1, but p_2 exchanges it with another one. In this case, we can restore the old order by applying p_2 once again to P_2, resulting in the graph K. Applying p_1 with the match m_1 leads to a common object. Altogether, we obtain the transformations $P_1 \overset{id}{\Rightarrow} P_1$ and $P_2 \overset{exchangeParameter}{\Longrightarrow} K \overset{p_1,m_1}{\Longrightarrow} P_1$.
5. $p_1 = p_2 = exchangeParameter$. Exchanging parameters can be reversed, so there are transformations $P_1 \overset{p_2}{\Longrightarrow} K$ and $P_2 \overset{p_1}{\Longrightarrow} K$.

In all these cases, the common part of K that is preserved by applying p_1 and p_2 is also preserved by the further transformations and mapped equally to the resulting common object. Therefore the critical pairs are strictly confluent.

By Theorem 10.18, this means that the graph grammar *MethodModeling* is locally confluent. □