Mechanised DPO Theory: Uniqueness of Derivations and Church-Rosser Theorem

Robert Söldner and Detlef Plump

Department of Computer Science, University of York, York, UK
{rs2040,detlef.plump}@york.ac.uk

Abstract. We demonstrate how to use the proof assistant Isabelle/HOL to obtain machine-checked proofs of two fundamental theorems in the double-pushout approach to graph transformation: the uniqueness of derivations up to isomorphism and the so-called Church-Rosser theorem. The first result involves proving the uniqueness of pushout complements, first established by Rosen in 1975. The second result formalises Ehrig's and Kreowski's proof of 1976 that parallelly independent direct derivations can be interchanged to obtain derivations ending in a common graph. We also show how to overcome Isabelle's limitation that graphs in locales must have nodes and edges of pre-defined types.

Keywords: Double-pushout graph transformation · Isabelle/HOL · Uniqueness of derivations · Church-Rosser theorem

1 Introduction

Formal methods help to mitigate the risk of software defects by rigorously verifying the correctness of software against their specification. It involves the use of mathematical techniques to prove that a software implementation meets a set of specifications and requirements. These methods include techniques such as model checking, theorem proving, and symbolic interpretation [13].

Interactive theorem provers have been used to rigorously prove mathematical theorems such as the Four Colour Theorem [9], the Prime Number Theorem [1], and the Kepler Conjecture [11]. Moreover, specific algorithms and software components have been successfully verified against their specifications, such as the seL4 Microkernel Verification [14] and the development and formal verification of the CompCert compiler [16]. These achievements demonstrate the potential of interactive theorem provers in verifying complex theories and systems.

Our research aims to rigorously prove fundamental results in the double-pushout approach to graph transformations [6], which aligns with our long-term goal of verifying specific programs in the graph programming language GP 2 [21,4] using the Isabelle proof assistant [18]. This paper extends earlier work [24] and builds the foundations for our long-term goal.

While we are not the first to apply interactive theorem provers to graph transformation, previous approaches such as Strecker's [25] do not address the vast amount of theoretical results available for the double-pushout approach, see

Section 6 for details. To the best of our knowledge, we are the first to formalise basic results in double-pushout graph transformation.

As usual in the double-pushout approach, we want to abstract from node and edge identifiers. Therefore, we introduce type variables, for each graph independently, that denote the type of the nodes and edges. Using different type variables for different graphs allows to use Isabelle's typechecker during development to prevent accidental mixing of node or edge identifiers from different graphs. However, a problem then is that Isabelle does not allow to quantify over new type variables within locale definitions. This poses a challenge to formalise the universal property of pushouts and pullbacks. We overcome this problem by requiring, in locales only, that nodes and edges are natural numbers.

To prove the uniqueness of direct derivations, we break down the proof into two parts. Firstly, we show the uniqueness of the pushout complement, and subsequently show that the pushout object is also unique given an isomorphic pushout complement. In the first part we follow Lack and Sobocinski's proof for adhesive categories [15], but replace adhesiveness with the pushout characterisation by the reduced chain condition [8] together with composition and decomposition lemmata for pushouts and pullbacks.

To prove the Church-Rosser theorem, we follow the original proof of Ehrig and Kreowski [7] and rely again on the pushout characterisation by the reduced chain condition. Additionally, we exploit composition and decomposition lemmata for pushouts and pullbacks, and also a lemma that allows to switch from pullbacks to pushouts and vice versa.

We believe that our proofs could be generalised from graphs to adhesive categories, but this is not our goal. Firstly, we want to avoid the overhead necessary to deal with an abstract class of categories, such as van Kampen squares. Secondly, our development addresses both abstract concepts such as pushouts and pullbacks, and set-theoretic constructions for such concepts. Giving corresponding constructions for all adhesive categories would not be feasible. Our goal is to lay the foundations for verifying graph programs in the language GP 2, a language whose syntax and semantics are built upon the double-pushout approach to graph transformation. Our ultimate goal is to provide both interactive and automatic tool support for formal reasoning on programs in graph transformation languages such as GP 2. An implementation of a proof assistant for GP 2 will have to use the concrete definitions of graphs, attributes, rules, derivations etc. Hence our formalisation focuses on concepts for transforming graphs. We are not saying that formalising the theory of adhesive categories is not interesting, but a large part of such an development would be a distraction from our main goal.

The rest of the paper is structured as follows. The next section is a brief introduction to Isabelle/HOL and the constructs used in our work. Section 3 describes the basics of DPO graph transformation and our formalisation in Isabelle. We show the uniqueness of direct derivations in Section 4, which entails a proof of the uniqueness of pushout complements. Section 5 introduces the so-called Church-Rosser theorem which states that parallelly independent direct derivations can be interchanged to obtain derivations ending in a common graph. Section 6 provides a brief overview of related work. Finally, in Section 7, we summarise our findings and discuss future work.

2 Isabelle/HOL

Isabelle is a generic, interactive theorem prover based on the so-called *LCF* approach. It is based on a small (meta-logical) proof kernel, which is responsible for checking all proofs. This concept provides high confidence in the prover's soundness. Isabelle/HOL refers to the higher-order logic instantiation, which is considered to be the most established calculus within the Isabelle distribution [20]. HOL is strongly typed with support for polymorphic, higher-order functions [3].

In Isabelle/HOL, type variables are denoted by a leading apostrophe. For example, a term f of type 'a is denoted by f :: 'a. Our formalisation is based on *locales*, a mechanism for writing and structuring parametric specifications. The locale's context comprises a set of parameters $(x_1 \dots x_n)$, assumptions $(A_1 \dots A_m)$ and concluding theorem $\bigwedge x_1 \dots x_n.\ [\![A_1; \dots ; A_m]\!] \Longrightarrow C$. This logical view offers a compelling way of combining and enhancing contexts, resulting in a clear and maintainable representation. A detailed introduction can be found in [2].

The special syntax $[\![\mathrm{A}_1;\ \dots;\ \mathrm{A}_n\ ]\!] \Longrightarrow \mathrm{P}$ is syntactic sugar for $\mathrm{A}_1 \Longrightarrow \dots \Longrightarrow \mathrm{A}_n \Longrightarrow \mathrm{C}$ and can be read as: if A_1 and … and A_n hold, C is implied. Furthermore, we use *intelligible semi-automated reasoning* (Isar) which is Isabelle‘s language of writing structured proofs [26]. In contrast to *apply-scripts*, which execute deduction rules in a linear manner, Isar follows a structured approach resulting in increased readability and maintainability [18]. A general introduction to Isabelle/HOL can be found in [18].

3 DPO Graph Transformation in Isabelle

We work with *directed labelled graphs* where the *label alphabet* consists of a set of node labels and a set of edge labels from which a graph can be labelled. In our Isabelle/HOL formalisation, we do not specify a concrete label alphabet and leave it unspecified. As a result, stated properties are satisfied for all possible label alphabets.

Our definition of graphs allows for parallel edges and loops, and we do not consider variables as labels.

Definition 1 (Graph). A *graph* $G = (V, E, s, t, l, m)$ over the alphabet $\mathcal{L}$ is a system where V is the finite set of nodes, E is the finite set of edges, $s, t \colon E \to V$ functions assigning the source and target to each edge, $l \colon V \to \mathcal{L}_V$ and $m \colon E \to \mathcal{L}_E$ are functions assigning a label to each node and edge. □

We represent this definition in Isabelle/HOL as a record type together with a locale. A discussion of this approach can be found in earlier work [24].

```
record ('v,'e,'l,'m) pre_graph =
  nodes  :: "'v set"
  edges  :: "'e set"
  source :: "'e ⇒ 'v"
  target :: "'e ⇒ 'v"
  node_label :: "'v ⇒ 'l"
  edge_label :: "'e ⇒ 'm"
```

In contrast to previous work [24], the locale required the built-in `countable` typeclass for the node and edge identifiers to overcome the limitation of Isabelle/HOL of introducing new type variables within the definition of a locale. This approach is similar to the one discussed in [23], in which the authors utilise Isabelle's `statespace` command to establish a local environment comprising axioms for the conversion of concrete types to natural numbers. Instead, we use the injective `to_nat` function through the use of the `countable` typeclass to facilitate the conversion of types.

```
locale graph =
  fixes G :: "('v::countable,'e::countable,'l,'m) pre_graph"
  assumes
    finite_nodes: "finite V_G" and
    finite_edges: "finite E_G" and
    source_integrity: "e ∈ E_G ⟹ s_G e ∈ V_G" and
    target_integrity: "e ∈ E_G ⟹ t_G e ∈ V_G"
```

The injectivity also implies the existence of the inverse, `from_nat`. We use this technique to translate arbitrary identifiers for nodes and edges into a generic representation based on natural numbers.

Definition 2 (Graphs over naturals ↗). A graph whose nodes and edges are natural numbers is called a *natural graph*.

In Isabelle/HOL, we create a type synonym `ngraph`, which specialises the existing `pre_graph` structure to `nat`, Isabelle's built-in type of natural numbers, for both identifiers.

```
type_synonym ('l,'m) ngraph = "(nat,nat,'l,'m) pre_graph"
```

The use of Isabelle/HOL's built-in functions `to_nat` and `from_nat` allows us to convert between both representations. We define the conversion function `to_ngraph` from `ngraph` to the parameterised `pre_graph` structure as follows:

```
definition to_ngraph
              :: "('v::countable,'e :: countable,'l,'m) pre_graph
              ⇒ ('l,'m) ngraph" where
 ‹to_ngraph G ≡ ⦇nodes  = to_nat ` V_G
                ,edges  = to_nat ` E_G
                ,source = λe. to_nat (s_G (from_nat e))
                ,target = λe. to_nat (t_G (from_nat e))
                ,node_label = λv. l_G (from_nat v)
                ,edge_label = λe. m_G (from_nat e)⦈›
```

The node and edge identifiers are mapped using `to_nat` while for the source (target) function, we first translate the natural number back into the parameterised, then apply the original source (target) function, and finally convert it back to a natural number.

A *graph morphism* $f\colon G \to H$ is a structure preserving mapping from a source graph G to a target graph H. It consists of a pair of mappings $f = (f_V\colon V_G \to V_H, f_E\colon V_E \to V_H)$ for nodes and edges, respectively. We say a morphism f is *injective* (*surjective*), if f_V and f_E are injective (surjective). A morphism which is injective and surjective is called *bijective.* We call G and H *isomorphic*, denoted by $G \cong H$, if a bijective morphism exists. The *morphism composition* of $f\colon F \to G$ and $g\colon G \to H$, which we denote by the usual infix notation $g \circ f$, is the pairwise composition of the node and edge mappings $g \circ f = (g_V \circ f_V, g_E \circ f_E)$. If a morphism is uniquely identified by its source and target, we sometimes omit the name and write $F \to G \to H$ to stand for the composition $g \circ f$. In our formalisation, we denote morphism composition using the $\circ_{\rightarrow}$ symbol to prevent a naming clash with Isabelle's built-in function composition. Different from our earlier investigation [24], we generalise *rules* to use injective morphisms instead of inclusions.

Definition 3 (Rule). A *rule* $(L \leftarrow K \to R)$ consists of graphs L, K and R over $\mathcal{L}$ together with injective morphisms $K \to L$ and $K \to R$. □

Another difference is the introduction of a new record type `pre_rule` to be consistent with our formalisation of graphs and morphisms. The left-hand, interface, and right-hand side of a rule can be accessed using the corresponding record accessor functions `lhs`, `interf`, and `rhs`. This approach reduces the amount of parameters we have to pass, resulting in better readability.

```
record ('v_1,'e_1,'v_2,'e_2,'v_3,'e_3,'l,'m) pre_rule =
  lhs    :: "('v_1,'e_1,'l,'m) pre_graph"
  interf :: "('v_2,'e_2,'l,'m) pre_graph"
  rhs    :: "('v_3,'e_3,'l,'m) pre_graph"
```

The `rule` locale now relies on the two injective morphisms $K \to L$ and $K \to R$. We populate the `countable` typeclass restriction to allow conversion of graphs to graphs over naturals. In order to leverage Isabelle's typechecker, each graph in a rule is allowed to have a different type for the node and edge identifiers.

```
locale rule =
  k: injective_morphism "interf r" "lhs r" b +
  r: injective_morphism "interf r" "rhs r" b'
  for r :: "('v_1::countable,'e_1::countable
           ,'v_2::countable,'e_2::countable
           ,'v_3::countable,'e_3::countable
           ,'l,'m) pre_rule" and b b'
```

In general, this design choice supports the development process, as the typechecker is rejecting certain terms. For example, we are not able to express membership of an item $x \in K_V \implies x \in L_V$ as the type of node identifiers in K is v_2 while L is of type v_1. From now on, we omit explicit type and typeclass restrictions within the shown code examples to enhance readability.

Definition 4 (Dangling condition). Let $b'\colon K \to L$ be an injective graph morphism. An injective graph morphism $g\colon L \to G$ satisfies the *dangling condition* if no edge in $E_G - g_E(E_L)$ is incident to a node in $g_V(V_L - b'_V(V_K))$. □

Definition 5 (Pushout). Given graph morphisms $b\colon A \to B$ and $c\colon A \to C$, a graph D together with graph morphisms $f\colon B \to D$ and $g\colon C \to D$ is a *pushout* of $A \to B$ and $A \to C$ if the following holds (see Fig. 1a):

1. Commutativity: $f \circ b = g \circ c$, and
2. Universal property: For all graph morphisms $p\colon B \to H$ and $t\colon C \to H$ such that $p \circ b = t \circ c$, there is a unique morphism $u\colon D \to H$ such that $u \circ f = p$ and $u \circ g = t$.

We call D the *pushout object* and C the *pushout complement.* □

Different from earlier work [24], we use graphs over naturals to formalise the universal property in order to overcome the mentioned limitation of Isabelle's locale mechanism. An important property is that pushouts are unique up to isomorphism, as formalised and proved in [24].

$$\begin{tikzcd}
& B \arrow[dr, "f"] \arrow[drr, bend left, "p"] & & \\
A \arrow[ur, "b"] \arrow[dr, "c"'] & & D \arrow[r, dashed] & H \\
& C \arrow[ur, "g"'] \arrow[urr, bend right, "t"'] & &
\end{tikzcd}$$

(a) Pushout $A \to B \to D \leftarrow C \leftarrow A$

$$\begin{tikzcd}
& & B \arrow[dr, "f"] & \\
H \arrow[r, dashed] \arrow[urr, bend left, "p"] \arrow[drr, bend right, "t"'] & A \arrow[ur, "b"] \arrow[dr, "c"'] & & D \\
& & C \arrow[ur, "g"'] &
\end{tikzcd}$$

(b) Pullback $D \leftarrow C \leftarrow A \to B \to D$

Fig. 1: Pushout and Pullback diagrams

Theorem 1 (Uniqueness of pushouts [6]). *Let $b\colon A \to B$ and $c\colon A \to C$ together with D induce a pushout as depicted in Fig. 1a. A graph H together with morphisms $p\colon B \to H$ and $t\colon C \to H$ is a pushout of b and c if and only if there is an isomorphism $u\colon D \to H$ such that $u \circ f = p$ and $u \circ g = t$.* □

The transformation of graphs by rules gives rise to *direct derivations.*

Definition 6 (Direct derivation). Let G and H be graphs, $r = \langle L \leftarrow K \to R \rangle$ and $g\colon L \to G$ an injective morphism satisfying the dangling condition. Then G *directly derives* H by r and g, denoted by $G \Rightarrow_{r,g} H$, as depicted in the double pushout diagram in Fig. 2. □

Note that the injectivity of the matching morphism $g\colon L \to K$ leads to a DPO approach that is more expressive than in the case of arbitrary matches [10].

In our earlier work [24], we used the `direct_derivation` locale to represent the operational view using gluing and deletion. Instead, here we use the categorical definition relying on pushouts. The operational definition is available within the `direct_derivation_construction` locale. We do not repeat its definition here, as it is not relevant for the current discussion.

```
locale direct_derivation =
  r: rule r b b' +
  gi: injective_morphism "lhs r" G g  +
  po1: pushout_diagram "interf r" "lhs r" D G b d g c +
  po2: pushout_diagram "interf r" "rhs r" D H b' d f c'
  for r b b' G g D d c H f c'
begin
```

$$\begin{tikzcd}
L \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & K \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(2)"] & R \arrow[d] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

Fig. 2: Double-pushout diagram

$$\begin{tikzcd}
A \arrow[r] \arrow[d] \arrow[dr, phantom, "(1)"] & B \arrow[r] \arrow[d] \arrow[dr, phantom, "(2)"] & E \arrow[d] \\
C \arrow[r] & D \arrow[r] & F
\end{tikzcd}$$

Fig. 3: Composite commutative diagram

A pullback is dual to the concept of pushout.

Definition 7 (Pullback). Given graph morphism $f\colon B \to D$ and $g\colon C \to D$, a graph A together with graph morphisms $b\colon A \to B$ and $c\colon A \to C$ is a *pullback* of $C \to D \leftarrow B$ if the following holds (see Fig. 1b):

1. Commutativity: $f \circ b = g \circ c$, and
2. Universal property: For all graph morphisms $p\colon H \to B$ and $t\colon H \to C$ such that $f \circ p = g \circ t$, there is a unique morphism $u\colon H \to A$ such that $b \circ u = p$ and $c \circ u = t$. □

Definition 8 (Pullback construction [6]). Let $f\colon B \to D$ and $g\colon C \to D$ be graph morphisms. Then the following defines a graph A (see Fig. 1b), the pullback object of f and g:

1. $A = \{\langle x, y\rangle \in B \times C \mid f(x) = g(y)\}$ for nodes and edges, respectively
2. $s_A(\langle x, y\rangle) = \langle s_B(x), s_c(y)\rangle$ for $\langle x, y\rangle \in E_B \times E_C$
3. $l_A(\langle x, y\rangle) = l_B(x)$ for $\langle x, y\rangle \in V_B \times V_C$
4. $b\colon A \to B$ and $c\colon A \to C$ are defined by $b(\langle x, y\rangle) = x$ and $c(\langle x, y\rangle) = y$ □

We formalise the pullback construction in the `pullback_construction` locale, assuming morphisms $f\colon B \to D$ and $g\colon C \to D$. Our definition of the pullback object A is as follows:

```
abbreviation V where
  <V ≡ {(x,y). x ∈ V_B ∧ y ∈ V_C ∧ f_V x = g_V y}>
abbreviation E where
  <E ≡ {(x,y). x ∈ E_B ∧ y ∈ E_C ∧ f_E x = g_E y}>
fun s where <s (x,y) = (s_B x, s_C y)>
fun t where <t (x,y) = (t_B x, t_C y)>
fun l where <l (x,_) = l_B x>
fun m where <m (x,_) = m_B x>

definition A where
  <A ≡ ⦇nodes = V, edges = E, source = s, target = t
       ,node_label = l, edge_label = m⦈>
```

The next lemma shows that this construction leads to a graph that is indeed the pullback object of Fig. 1b.

Lemma 1 (Correctness of pullback construction). *Let $f\colon B \to D$ and $g\colon C \to D$ be graph morphisms and let graph A and graph morphisms b and c be defined as in Def. 8. Then the square in Fig. 1b is a pullback diagram.*

We use the `sublocale` command, instead of `interpretation`, to make these facts persistent in the current context via the `pb` identifier.

```
sublocale pb: pullback_diagram A B C D b c f g
```

Similar to pushouts, pullbacks are unique up to isomorphism.

Theorem 2 (Uniqueness of pullbacks). *Let $f\colon B \to D$ and $g\colon C \to D$ together with A induce a pullback as depicted in Fig. 1b. A graph H together with morphisms $p\colon H \to B$ and $t\colon H \to C$ is a pullback of f and g if and only if there is an isomorphism $u\colon H \to U$ such that $b \circ u = p$ and $c \circ u = t$.*

The proof is dual to the uniqueness of the pushout (cf. Theorem 1) and we omit it due to space limitations. Essential properties for the forthcoming proofs in Section 4 and Section 5 are the composition and decomposition of pushouts and pullbacks.

Lemma 2 (Pushout/Pullback composition and decomposition). *Given the commutative diagram in Fig. 3, then the following statements are true:*

1. *If* (1) *and* (2) *are pushouts, so is* $(1) + (2)$
2. *If* (1) *and* $(1) + (2)$ *are pushouts, so is* (2)
3. *If* (1) *and* (2) *are pullbacks, so is* $(1) + (2)$
4. *If* (2) *and* $(1) + (2)$ *are pullbacks, so is* (1) □

For example, item 3 of Lemma 2 is expressed in Isabelle/HOL as follows:

```
lemma pullback_composition:
  assumes
    1: ‹pullback_diagram A B C D f g g' f'› and
    2: ‹pullback_diagram B E D F e g' e'' e'›
  shows ‹pullback_diagram A E C F (e ∘→ f) g e'' (e' ∘→ f')›
```

The proof is analogous to [6, Fact 2.27]. In the next section, we show that direct derivations have a unique (up to isomorphism) result.

4 Uniqueness of Direct Derivations

The uniqueness of direct derivations is an important property when reasoning about rule applications. This section does not rely on the adhesiveness of the category of graphs, instead we base our proof on the characterisation of graph pushouts in [8]. Before stating the theorem, we introduce additional facts, mainly about pushouts and pullbacks, which are used within the proof of Theorem 4.

In general, pushouts along injective morphisms are also pullbacks.

$$\begin{tikzcd}
 & & K \arrow[dll, "id"'] \arrow[dr, "h"] \arrow[ddd, "id"] & \\
K \arrow[dr, "d'"] \arrow[ddd, "b"'] & & & U \arrow[dll, "k"] \arrow[ddd, "l"] \\
 & D' \arrow[ddd, "m"'] & & \\
 & & K \arrow[dll, "b"'] \arrow[dr, "d"] & \\
L \arrow[dr, "g"'] & & & D \arrow[dll, "c"] \\
 & G & &
\end{tikzcd}$$

Fig. 4: Commutative cube based on direct derivation [15]

Lemma 3 (Injective pushouts are pullbacks [6] ↗). *A pushout diagram as depicted in Fig. 1a is also a pullback if $A \to B$ is injective.*

The proof relies on the pullback construction (cf. Def. 8) and the fact that pullbacks are unique (cf. Theorem 2). Furthermore, pushouts and pullbacks preserve injectivity (surjectivity) in the sense that the opposite morphisms of the corresponding diagram (see Fig. 1) is also injective (surjective).

Lemma 4 (Preservation of injective and surjective morphisms [6] ↗). *Given a pushout diagram in Fig. 1a, if $A \to B$ is injective (surjective), so is $C \to D$. Given a pullback diagram in Fig. 1b, if $C \to D$ is injective (surjective), so is $A \to B$.*

Certain forms of commutative diagrams give raise to pullbacks. This property is used in the proof of the uniqueness of the pushout complement (cf. Theorem 4).

Lemma 5 (Special pullbacks [6] ↗). *The commutative diagram in Fig. 6 is a pullback if m is injective.*

Definition 9 (Reduced chain-condition [8]). The commutative diagram in Fig. 5 satisfies the *reduced chain-condition*, if for all $b' \in B$ and $c' \in C$ with $f(b') = g(c')$ there is $a \in A$ such that $b(a) = b'$ and $c(a) = c'$.

We show that pullbacks satisfy the reduced chain-condition.

Lemma 6 (Pullbacks satisfy the reduced chain-condition ↗). *Each pullback diagram as depicted in Fig. 1b satisfies the reduced chain-condition.*

We state this lemma in Isabelle as follows:

```
lemma reduced_chain_condition_nodes:
  fixes x y
  assumes ‹x ∈ V_B› ‹y ∈ V_C› ‹f_V x = g_V y›
  shows ‹∃a ∈ V_A. (b_V a = x ∧ c_V a = y)›
```

Our proof relies on the pullback construction (cf. Def. 8) and the fact, that pullbacks are unique (cf. Theorem 2).

Definition 10 (Jointly surjective). Given injective graph morphisms $f\colon B \to D$ and $g\colon C \to D$. f and g are *jointly surjective*, if each item in D has a preimage in B or C.

Theorem 3 (Pushout characterization [8]). *The commutative diagram in Fig. 5 is a pushout, if the following conditions are true:*

1. *The morphisms b, c, f, g are injective.*
2. *The diagram satisfies the reduced chain-condition.*
3. *The morphisms g, f are jointly surjective.*

$$\begin{tikzcd}
 & A \arrow[dl, "c"'] \arrow[dr, "b"] & \\
C \arrow[dr, "g"'] & & B \arrow[dl, "f"] \\
 & D &
\end{tikzcd}$$

Fig. 5: Commutative diagram

$$\begin{tikzcd}
 & & C \arrow[dr, "m"] & \\
H \arrow[r, dashed] \arrow[urr, bend left] \arrow[drr, bend right] & C \arrow[ur, "id"] \arrow[dr, "id"'] & & D \\
 & & C \arrow[ur, "m"'] &
\end{tikzcd}$$

Fig. 6: Special pullback diagram

The following theorem implies the uniqueness of the pushout complement, which is known to hold if the morphism $K \to L$ in the applied rule is injective, even if the matching morphism $L \to G$ is non-injective [22]. In our case, both morphisms are injective.

Theorem 4 (Uniqueness of direct derivations). *Let* (1)+(2) *and* (3)+(4) *be direct derivations as depicted in Fig. 9. Then $D \cong D'$ and $H \cong H'$.*

This theorem is stated in Isabelle/HOL within the locale `direct_derivation`. Within the `assumes` part, the second direct derivation, which we call `dd2`, is introduced.

```
theorem uniqueness_direct_derivation:
  assumes
    dd2: <direct_derivation r b b' G g D' d' m H' f' m'>
  shows <(∃u. bijective_morphism D D' u) ∧
         (∃u. bijective_morphism H H' u)>
```

The uniqueness proof of direct derivations (see Fig.9) is performed in two phases. Firstly, we show the uniqueness of the pushout complement, which was first shown by Rosen [22]. Subsequently, we show that given a bijection between D and D', the pushout object is also unique up to isomorphism.

Proof (Theorem 4). The first phase of our proof closely follows Lack and Sobocinski [15], except for the final step, where the authors rely on adhesiveness. We finish the proof by relying on the pushout characterization (cf. Theorem 3). Given the two pushout diagrams (1) and (3) in Fig. 9 with injective $K \to D$ and $K \to D'$. To show the existence of a bijection between D and D', we construct the commutative cube in Fig. 4 with (1) as the bottom face and (3) as the front-left face, and show that l and k are bijections. For the latter, we show that the back-right and top faces are pushouts. (In [15], this is shown by adhesiveness, while we argue with the pushout characterization of Theorem 3.) The front-right face is a pullback construction (cf. Def. 8) which we tell Isabelle by interpretation of the `pullback_construction` locale.

```
interpret fr: pullback_construction D G D' c m ..
```

We use Isabelle's shorthand notation `..` for the `standard` tactic, to discharge the proof obligations which follow from the assumptions. Note that the pullback object together with the two morphisms is specified within the locale. Subsequent code will reference the pullback object by `fr.A`, the morphism l by `fr.b` and k by `fr.c` (see Fig. 4). (The identifiers within locales are given by the definition. As a result, the pullback object of the front-right face is referred to as `A` rather than the interpretation parameter `K`.) From Lemma 5, in our formalization referenced by `fun_algrtr_4_7_2`, we know that the back-left face is a pullback.

```
interpret bl: pullback_diagram "interf r" "interf r"
                "interf r" "lhs r" idM idM b b
  using fun_algrtr_4_7_2[OF r.k.injective_morphism_axioms]
  by assumption
```

To show that the back-right face is a pullback, we start with the front-left face. As the front left face is a pushout and m is injective, n' is too (cf. Lemma 4). Since pushouts along injective morphisms are also pullbacks (cf. Lemma 3), the front-left face is also a pullback. Using the pullback composition (cf. Lemma 2), the back face is pullback.

```
interpret backside: pullback_diagram "interf r" D' "interf r" G
                      <d' ∘→ idM> idM m <g ∘→ b>
  using pullback_composition[OF bl.pullback_diagram_axioms
                                dd2.pb1.flip_diagram]
  by assumption
```

We define $h\colon K \to U$ using both, the d and d' morphisms as $h\ x = (d\ x, d'\ x)$ and subsequently prove the morphism properties.

```
  define h where
    <h ≡ ⦇node_map = λv. (d_V v, d'_V v)
         ,edge_map = λe. (d_E e, d'_E e)⦈>
```

We follow by showing that the top and bottom face commutes, i.e., $d' \circ id = k \circ h$ and $g \circ b = c \circ d$, respectively. This establishes the fact, that the right-side of the cube is a pullback. Using the pullback decomposition (cf. Lemma 2), the back-right face is a pullback. To approach the top-face, we start by showing it is a pullback, and subsequently it is also a pushout. Since m is injective, from Lemma 3 we know the bottom-face is also a pullback. Using the pullback composition (cf. Lemma 2), the bottom and back-left face is a pullback. By commutativity of the bottom face $g \circ b = c \circ d$ and back-right face $l \circ h = d \circ id$, the front-right and top face is a pullback. By the pullback decomposition (cf. Lemma 2) we can show that the top face is a pullback. We show this pullback is also a pushout by using the pushout characterization (cf. Theorem 3). Therefore, we need to show that h is injective, which follows from the construction above of h.

```
interpret h: injective_morphism "interf r" fr.A h
proof
  show <inj_on hV Vinterf r>
    using d_inj.inj_nodes
    by (simp add: h_def inj_on_def)
next
  show <inj_on hE Einterf r>
    using d_inj.inj_edges
    by (simp add: h_def inj_on_def)
qed
```

Joint surjectivity of k and d' follows from the pullback construction and the reduced-chain condition (cf. Lemma 9) of the front-left and top face. Note, that the reduced-chain condition holds for all pullbacks. Finally, we need to show that k and l are bijections. Since the top face is a pushout and the $C \to C$ morphism is a bijection, by Lemma 4, so is k.

```
interpret k_bij: bijective_morphism fr.A D' fr.c
  using top.b_bij_imp_g_bij[OF r.k.G.idm.bijective_morphism_axioms]
  by assumption
```

To show l is a bijection, we show that the back-right face is a pushout by using the pushout characterization. The bijectivity of l follows from the fact pushouts preserve bijections. We follow by defining the morphism $u\colon D \to D'$ as the composition of the l^{-1} and k. The inverse of l is obtained by using a lemma within our formalization of bijective morphisms, stating the existence of the inverse (`ex_inv`) using the `obtain` keyword:

```
obtain linv where linv:<bijective_morphism D fr.A linv>
  and <⋀v. v ∈ VD⟹ fr.b ∘→linvV v = v>
      <⋀e. e ∈ ED⟹ fr.b ∘→linvE e = e>
  and <⋀v. v ∈ Vfr.A⟹ linv ∘→ fr.b V v = v>
      <⋀e. e ∈ Efr.A⟹ linv ∘→ fr.b E e= e>
  by (metis l_bij.ex_inv)
```

We finish the first phase by defining the morphism $u\colon D \to D'$ as $k \circ l^{-1}$ and subsequently prove that morphism composition preserves bijections (using the already proven `bij_comp_is_bij` lemma).

```
define u where <u ≡ fr.c ∘→ linv>

interpret u: bijective_morphism D D' u
  using bij_comp_bij_is_bij[OF linv k_bij.bijective_morphism_axioms]
  by (simp add: u_def)
```

$$\begin{tikzcd}
& R \arrow[dr, "f"] \arrow[drr, bend left, "f'"] & & \\
K \arrow[ur, "b'"] \arrow[dr, "d"'] & & H \arrow[r, dashed, "u'"] & H' \\
& D \arrow[ur, "c'"'] \arrow[urr, bend right, "m' \circ u"'] & &
\end{tikzcd}$$

Fig. 7: Construction of u'

$$\begin{tikzcd}
& R \arrow[dr, "f'"] \arrow[drr, bend left, "f"] & & \\
K \arrow[ur, "b'"] \arrow[dr, "d'"'] & & H' \arrow[r, dashed, "u''"] & H \\
& D' \arrow[ur, "m'"'] \arrow[urr, bend right, "c' \circ u^{-1}"'] & &
\end{tikzcd}$$

Fig. 8: Construction of u''

The second phase is to show the existence of an isomorphism $H \to H'$. We start by obtaining $u' \colon H \to H'$ and $u'' \colon H' \to H$ and show they are inverses. We use the universal property of the pushout depicted in Fig. 7, which requires us to show commutativity: $f' \circ b' = m' \circ u \circ d$. So we substitute $u \circ d = d'$ into the commutativity equation of pushout (4) ($f' \circ b' = m' \circ d'$) in Fig. 9. We get $u \circ d = d'$ as follows: $u \circ d \overset{(1)}{=} k \circ l^{-1} \circ d \overset{(2)}{=} k \circ l^{-1} \circ l \circ h \overset{(3)}{=} k \circ h \overset{(4)}{=} d'$. Here, (1) is justified by the definition of u, (2) by the definition of l and h (which makes the back-right face in Fig. 4 commute), (3) by inverse cancellation, and finally (4) by the definitions of k and h (similarly to step (2)). We obtain $u'' \colon H' \to H$ by using the universal property of the pushout depicted in Fig. 8. We show the commutativity $f \circ b = c' \circ u^{-1} \circ d'$ by substituting $u^{-1} \circ d' = d$ into the commutativity equation of pushout (2) ($f \circ b' = c' \circ d$) in Fig. 9: $u^{-1} \circ d' \overset{(5)}{=} u^{-1} \circ u \circ d \overset{(6)}{=} d$. Here, (5) is justified by the above proven equation $u \circ d = d'$, and (6) follows from cancellation of inverses. The final steps are to show $u' \circ u'' = id$ and $u'' \circ u' = id$. To show the first equation, we start with $f' = u' \circ u'' \circ f'$ and $m' = u' \circ u'' \circ m'$, which we get from the definitions of u' and u''. Using the universal property of pushout (4) in Fig. 9 together with H', f', m', we conclude that the identity is the unique morphism $H' \to H'$ that makes the triangles commute. If $u' \circ u''$ makes the triangles commute as well, it is equal to the identity morphism. The first triangle commutes because $u' \circ u'' \circ f' \overset{(7)}{=} u' \circ f \overset{(8)}{=} f'$. Here, (7) and (8) are justified by the corresponding construction of u' and u'' (see the triangles in Fig. 7 and Fig. 8). For the second triangle, we start by using the commutativity of the bottom triangle in Fig. 7 and composing to the right with u^{-1}: $u' \circ c' \circ u^{-1} = m' \circ u \circ u^{-1}$. By cancellation of inverses we get $u' \circ c' \circ u^{-1} = m'$ and by substituting $c' \circ u^{-1}$ using the commutativity of the bottom triangle in Fig. 8, we prove that $u' \circ u'' \circ m' = m'$. Showing that $u'' \circ u' = id$ follows analogously and is omitted to save space. □

With the uniqueness of direct derivations (cf. Theorem 4), we get the uniqueness of the pushout complement.

Corollary 1 (Uniqueness of pushout complements ↗). *Given a pushout as depicted in Fig. 1a where $A \to B$ is injective. Then the graph D is unique up to isomorphism.*

$$\begin{tikzcd}
L \arrow[d, "g"'] \arrow[dd, bend left, "g"] \arrow[r, phantom, ""] & K \arrow[l] \arrow[r] \arrow[d] \arrow[dd, bend left] & R \arrow[d] \arrow[dd, bend left] \\
G \arrow[r, phantom, ""] & D \arrow[l] \arrow[r] & H \\
G & D' \arrow[l] \arrow[r] & H'
\end{tikzcd}$$

(1) (2) (3) (4)

Fig. 9: Uniqueness of direct derivations

$$\begin{tikzcd}
& G \arrow[dl, Rightarrow, "p_1, m_1"'] \arrow[dr, Rightarrow, "p_2, m_2"] & \\
H_1 \arrow[dr, Rightarrow, "p_2, m_{2'}"'] & & H_2 \arrow[dl, Rightarrow, "p_1, m_{1'}"] \\
& G' &
\end{tikzcd}$$

Fig. 10: Church-Rosser Theorem

We omit the proof to conserve space. The upcoming section introduces the so-called Church-Rosser Theorem, which states that parallell independent direct derivations have the diamond property.

5 Church-Rosser Theorem

The Church-Rosser Theorem refers to the idea that two graph transformation rules can be applied independently of each other, either sequentially or in parallel, without changing the final result. We follow the independence characterization of direct derivations given in [6].

Definition 11 (Parallel independence [6] ↗). The two direct derivations $G \Rightarrow_{p_1,m_1} H_1$ and $G \Rightarrow_{p_2,m_2} H_2$ in Fig. 11 are *parallel independent* if there exists morphisms $L_1 \to D_2$ and $L_{\to}D_1$ such that $L_1 \to D_2 \to G = L_1 \to G$ and $L_2 \to D_1 \to G = L_2 \to G$. □

$$\begin{tikzcd}
R_1 \arrow[d] & K_1 \arrow[l] \arrow[r] \arrow[d] & L_1 \arrow[dr, "m_1"] \arrow[drrr, dashed] & & L_2 \arrow[dl, "m_2"'] \arrow[dll, dashed] & K_2 \arrow[l] \arrow[r] \arrow[d] & R_2 \arrow[d] \\
H_1 & D_1 \arrow[l] \arrow[rr] & & G & & D_2 \arrow[ll] \arrow[r] & H_2
\end{tikzcd}$$

(2) (1) (3) (4)

Fig. 11: Parallel independence

```
locale parallel_independence =
  p₁: direct_derivation r₁ b₁ b₁' G g₁ D₁ m₁ c₁ H₁ f₁ h₁ +
  p₂: direct_derivation r₂ b₂ b₂' G g₂ D₂ m₂ c₂ H₂ f₂ h₂
  for r₁ b₁ b₁' G g₁ D₁ m₁ c₁ H₁ f₁ h₁
      r₂ b₂ b₂' g₂ D₂ m₂ c₂ H₂ f₂ h₂ +
  assumes
    i: ‹∃i. morphism (lhs r₁) D₂ i
            ∧ (∀v ∈ V_lhs r₁. c₂ ∘→ i^V v = g₁V v)
            ∧ (∀e ∈ E_lhs r₁. c₂ ∘→ i^E e = g₁E e)› and
    j: ‹∃j. morphism (lhs r₂) D₁ j
            ∧ (∀v ∈ V_lhs r₂. c₁ ∘→ j^V v = g₂V v)
            ∧ (∀e ∈ E_lhs r₂. c₁ ∘→ j^E e = g₂E e)›
```

Theorem 5 (Church-Rosser Theorem [7] ↗). *Given two parallel independent direct derivations $G \Rightarrow_{p_1,m_1} H_1$ and $G \Rightarrow_{p_2,m_2} H_2$, there is a graph G' together with direct derivations $H_1 \Rightarrow_{p_2,m_2'} G'$ and $H_2 \Rightarrow_{p_1,m_1'} G'$.*

Actually, we have shown more, namely that $G \Rightarrow_{p_1,m_1} H_1 \Rightarrow_{p_2,m_2'} G'$ and $G \Rightarrow_{p_2,m_2} H_2 \Rightarrow_{p_1,m_1'} G'$ are sequentially independent. We express this theorem in Isabelle/HOL within the `parallel_independence` locale as follows:

```
theorem (in parallel_independence) church_rosser:
  shows <∃g' D' m' c' H f' h' g'' D'' m'' c'' H f'' h''.
           direct_derivation r₂ b₂ b₂' H₂ g'  D'  m'  c'  H f'  h'
         ∧ direct_derivation r₁ b₁ b₁' H₁ g'' D'' m'' c'' H f'' h''>
```

Proof (Theorem 5). We closely follow the original proof by Ehrig and Kreowksi [7] where in a first stage, the pushouts (1) - (4) of Fig. 11 are vertically decomposed into pushouts $(11) + (11)$, $(21) + (22)$, $(31) + (32)$, and $(41) + (42)$, as depicted in Fig. 12. In a second stage, these pushouts are rearranged as in Fig. 13 and the new pushout (5) is constructed. Subsequently, we prove the two vertical pushouts (11) and (12). The pushouts (31) and (32) follow analogously and are not shown to conserve space.

We start by constructing the pullback (12) which we bind to the symbol `c12`, allowing later references, using our `pullback_construction` locale.

```
  interpret "c12": pullback_construction D₁ G D₂ c₁ c₂ ..
```

The existence of $K_1 \to D$ follows from the universal property, and $D \to D_2$ from the construction of the pullback (12):

```
  obtain j₁ where <morphism (interf r₁) c12.A j₁>
    and <⋀v. v∈V_interf r₁ ⟹ c12.b ∘→ j₁V v = m₁V v>
        <⋀e. e∈E_interf r₁ ⟹ c12.b ∘→ j₁E e = m₁E e>
    and <⋀v. v∈V_interf r₁ ⟹ c12.c ∘→ j₁V v = i₁ ∘→ b₁V v>
        <⋀e. e∈E_interf r₁ ⟹ c12.c ∘→ j₁E e = i₁ ∘→ b₁E e>
    using c12.pb.universal_property_exist_gen[OF p₁.r.k.G.graph_axioms
              wf_b₁i₁.morphism_axioms p₁.po1.c.morphism_axioms a b]
    by fast
```

From the fact that $(1) = (11) + (12)$, we know $(11) + (12)$ is a pushout and since $K_1 \to L_1$ is injective, it is also a pullback (cf. Lemma 3). By pullback decomposition (cf. Lemma 2), (11) is a pullback. We use the pushout characterization (cf. Theorem 3) to show it is also a pushout, which requires us to show injectivity of all morphisms, reduced-chain condition, and joint surjectivity of $D \to D_2$ and $L_1 \to D_2$. The injectivity of $K_1 \to L_1$ is given, $D \to D_2$ follows from pushout (1) and the injectivity of $K_1 \to L_1$ (cf. Lemma 4). To show the injectivity of $L_1 \to D_2$, we use the parallel independence (cf. Def. 11) $L_1 \to D_2 \to G = L_1 \to G$ and the injectivity of $L_1 \to G$.To show injectivity of $K_1 \to D$, we use the triangle $L_1 \to D_2 \to G = L_1 \to G$ obtained by the universal property of pullback (12) and the injectivity of both, $L_1 \to G$ and $D_2 \to G$. The reduced-chain condition follows by Lemma 6. To show the joint surjectivity of $D \to D_2$ and $L_1 \to D_2$ (that is each x in D_2 has a preimage in either D or L_1). Let y be the image of x in G. We apply the joint surjectivity of pushout

(11) + (12) to y, that is y has a preimage in either D_1 or L_1. In the former case (y has a preimage z in D_1): from the pullback construction (cf. Def. 8), we get the common preimage of z and x in D which shows the former case. In the latter case, y has a preimage in L_1 via D_2. Since $D_2 \to G$ is injective, that preimage is mapped via x which means, x has a preimage in L_1. This shows the latter case.

$$\begin{tikzcd}
R_1 \arrow[d] \arrow[dd, bend right] \arrow[dr, phantom, "(21)"] & K_1 \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(11)"] & L_1 \arrow[d] \arrow[ddr, "m_1"] & & L_2 \arrow[d] \arrow[ddl, "m_2"'] \arrow[dr, phantom, "(31)"] & K_2 \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(41)"] & R_2 \arrow[d] \arrow[dd, bend left] \\
\overline{D}_2 \arrow[d] \arrow[dr, phantom, "(22)"] & D \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(12)"] & D_2 \arrow[dr] & & D_1 \arrow[dl] \arrow[dr, phantom, "(32)"] & D \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(42)"] & \overline{D}_1 \arrow[d] \\
H_1 & D_1 \arrow[l] \arrow[rr] & & G & & D_2 \arrow[ll] \arrow[r] & H_2
\end{tikzcd}$$

Fig. 12: Vertical pushout decomposition of Fig. 11

The pushouts (21), (41) are constructed using the `gluing` locale (see [24] for a detailed description).

```
interpret "c21": gluing "interf r₁" c12.A "rhs r₁" j₁ b₁' ..
interpret "c41": gluing "interf r₂" c12.A "rhs r₂" j₂ b₂' ..
```

The existence of $\overline{D}_2 \to H_1$ and $\overline{D}_1 \to H_2$ follows from the universal property of pushout (21) and (41), respectively. Pushouts (22) and (42) are obtained using the pushout decomposition (see Lemma 2). This finishes the first stage of the proof. The second stage rearranges the pushouts, as depicted in Fig. 13, such

$$\begin{tikzcd}
L_1 \arrow[d] \arrow[dd, bend right, "m_1"'] \arrow[dr, phantom, "(11)"] & K_1 \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(21)"] & R_1 \arrow[d] \arrow[ddr] & & L_2 \arrow[d] \arrow[ddl, "m_2'"'] \arrow[dr, phantom, "(31)"] & K_2 \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(41)"] & R_2 \arrow[d] \arrow[dd, bend left] \\
D_2 \arrow[d] \arrow[dr, phantom, "(12)"] & D \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(22)"] & \overline{D}_2 \arrow[dr] & & D_1 \arrow[dl] \arrow[dr, phantom, "(22)"] & D \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(5)"] & \overline{D}_1 \arrow[d] \\
G & D_1 \arrow[l] \arrow[rr] & & H_1 & & \overline{D}_2 \arrow[ll] \arrow[r] & G'
\end{tikzcd}$$

$$\begin{tikzcd}
L_2 \arrow[d] \arrow[dd, bend right, "m_2"'] \arrow[dr, phantom, "(31)"] & K_2 \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(41)"] & R_2 \arrow[d] \arrow[ddr] & & L_1 \arrow[d] \arrow[ddl, "m_1'"'] \arrow[dr, phantom, "(11)"] & K_1 \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(21)"] & R_1 \arrow[d] \arrow[dd, bend left] \\
D_1 \arrow[d] \arrow[dr, phantom, "(32)"] & D \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(42)"] & \overline{D}_1 \arrow[dr] & & D_2 \arrow[dl] \arrow[dr, phantom, "(42)"] & D \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(5)"] & \overline{D}_2 \arrow[d] \\
G & D_2 \arrow[l] \arrow[rr] & & H_2 & & \overline{D}_1 \arrow[ll] \arrow[r] & G'
\end{tikzcd}$$

Fig. 13: Rearranged pushouts of Fig. 12

that we obtain two direct derivations $H_1 \Rightarrow_{p_2,m'_2} G'$ and $H_2 \Rightarrow_{p_1,m'_1} G'$. Here, we compose the pushouts from stage 1 (see Lemma 2). Exemplary, the pushouts (31) and (22) are composed in Isabelle/HOL.

```
interpret "31+22": pushout_diagram "interf r_2" c21.H "lhs r_2" H_1
                                   "c21.c ∘→ j_2" b_2 s_1 "h_1 ∘→ i_2"
  using pushout_composition[OF
      "31.flip_diagram" "22.flip_diagram" ] by assumption
```

The final pushout (5) is constructed and the pushouts are rearranged and vertically composed as depicted in Fig. 13. Isabelle is able to discharge the goal at this point automatically as we instantiated all required locales. This finishes the second stage of the proof.

6 Related Work

Isabelle/HOL was used by Strecker for interactive reasoning about graph transformations [25]. A major difference to our work is that he introduces a form of graph transformation that does not fit with any of the established approaches such as the double-pushout approach. As a consequence, his framework cannot draw on existing theory. Another difference is that [25] focuses on verifying first-order properties of some form of graph programs while the current paper is concerned with formalising and proving fundamental results of the DPO theory. Strecker's formalisation fixes node and edge identifiers as natural numbers, while we keep them abstract. Similar to our development, Isabelle's locale mechanism is employed.

Our formalisation of graphs follows the work of Noschinski [19], where records are used to group components and locales to enforce properties such as the well-formedness of graphs or morphisms. The main objective of [19] is to formalise and prove fundamental results of classical graph theory, such as Kuratowski's theorem.

da Costa Cavalheiro et al. [5] use the Event-B specification method and its associated theorem prover to reason about double-pushout and single-pushout graph transformations, where rules can have attributes and negative application conditions. Event-B is based on first-order logic and typed set theory. Different from our approach, [5] gives only a non-formalised proof for the equivalence between the abstract definition of pushouts and the set-theoretic construction. In contrast, we formalise both the abstract and the operational view and prove their correspondence using Isabelle/HOL. As Event-B is based on first-order logic, the properties that can be expressed and verified are quite limited. For it is known that non-local properties of finite graphs cannot be specified in first-order logic [17]. This restriction does not apply to our formalisation as we can make full use of higher-order logic.

7 Conclusion

In this paper, we formalise and prove in Isabelle/HOL two fundamental results in the theory of double-pushout graph transformation, namely the uniqueness of direct derivations and the so-called Church-Rosser theorem. Furthermore, we describe an approach to overcome the restriction of introducing new type variables within locale definitions.

Drawing on our experience so far, we plan to simplify the formalisation by reducing the number of type variables needed. The idea is to employ a single global type variable for both node and edge identifiers within graphs and morphisms. As a consequence, each graph in our commutative diagrams (such as pushouts or pullbacks) would share this ID-type whereas currently, each occurring graph may have a different ID-type. This would significantly decrease the number of parameters in our formalisation, and would also eliminate the need to work around Isabelle's restriction on introducing new type variables within locales. To implement this idea, we will need to revise our pushout and pullback constructions.

Our next objective is to extend the current approach to encompass attributed DPO graph transformation in the sense of [12]. Ultimately, we want to build a GP 2 proof assistant within Isabelle/HOL that allows to interactively verify individual graph programs. Such a tool may use, for example, the proof calculus presented in [27].

Acknowledgements We are grateful to Brian Courthoute and Annegret Habel for discussions on the topics of this paper.

References

1. Avigad, J., Donnelly, K., Gray, D., Raff, P.: A formally verified proof of the prime number theorem. ACM Transactions on Computational Logic **9**(1), 2 (2007). https://doi.org/10.1145/1297658.1297660
2. Ballarin, C.: Tutorial to Locales and Locale Interpretation (2021), https://isabelle.in.tum.de/doc/locales.pdf
3. Brucker, A.D., Herzberg, M.: A Formal Semantics of the Core DOM in Isabelle/HOL. In: Companion Proceedings of the The Web Conference 2018. p. 741–749. WWW '18, International World Wide Web Conferences Steering Committee (2018). https://doi.org/10.1145/3184558.3185980
4. Campbell, G., Courtehoute, B., Plump, D.: Fast rule-based graph programs. Science of Computer Programming **214** (2022). https://doi.org/10.1016/j.scico.2021.102727
5. da Costa Cavalheiro, S.A., Foss, L., Ribeiro, L.: Theorem proving graph grammars with attributes and negative application conditions. Theoretical Computer Science **686**, 25–77 (2017). https://doi.org/10.1016/j.tcs.2017.04.010
6. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science, Springer (2006). https://doi.org/10.1007/3-540-31188-2
7. Ehrig, H., Kreowski, H.J.: Parallelism of manipulations in multidimensional information structures. In: Proc. Mathematical Foundations of Computer Science (MFCS 1976). Lecture Notes in Computer Science, vol. 45, pp. 284–293. Springer (1976)
8. Ehrig, H., Kreowski, H.J.: Pushout-properties: An analysis of gluing constructions for graphs. Mathematische Nachrichten **91**, 135–149 (1979)

9. Gonthier, G.: The four colour theorem: Engineering of a formal proof. In: Proc. Asian Symposium on Computer Mathematics (ASCN 2007). Lecture Notes in Computer Science, vol. 5081, p. 333. Springer (2007). https://doi.org/10.1007/978-3-540-87827-8_28
10. Habel, A., Müller, J., Plump, D.: Double-pushout graph transformation revisited. Mathematical Structures in Computer Science **11**(5), 637–688 (2001). https://doi.org/10.17/S0960129501003425
11. Hales, T., Adams, M., Bauer, G., Dang, D.T., Harrison, J., Hoang, T.L., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T.T., Nguyen, T.Q., Nipkow, T., Obua, S., Pleso, J., Rute, J., Solovyev, A., Ta, A.H.T., Tran, T.N., Trieu, D.T., Urban, J., Vu, K.K., Zumkeller, R.: A formal proof of the Kepler conjecture. Forum of Mathematics, Pi **5** (2015). https://doi.org/10.1017/fmp.2017.1
12. Hristakiev, I., Plump, D.: Attributed graph transformation via rule schemata: Church-Rosser theorem. In: Software Technologies: Applications and Foundations – STAF 2016 Collocated Workshops, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9946, pp. 145–160. Springer (2016). https://doi.org/10.1007/978-3-319-50230-4_11
13. Huth, M., Ryan, M.D.: Logic in Computer Science - Modelling and Reasoning about Systems. Cambridge University Press, 2nd edn. (2004)
14. Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D.A., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: formal verification of an OS kernel. In: Proc. Symposium on Operating Systems Principles (SOSP 2009). pp. 207–220. Association for Computing Machinery (2009). https://doi.org/10.1145/1629575.1629596
15. Lack, S., Sobociński, P.: Adhesive categories. In: Proc. Foundations of Software Science and Computation Structures (FoSSaCS 2004). Lecture Notes in Computer Science, vol. 2987, pp. 273–288. Springer (2004). https://doi.org/10.1007/978-3-540-24727-2_20
16. Leroy, X.: Formal verification of a realistic compiler. Communications of the ACM **52**(7), 107–115 (2009). https://doi.org/10.1145/1538788.1538814
17. Libkin, L.: Elements of Finite Model Theory. Texts in Theoretical Computer Science, Springer (2004). https://doi.org/10.1007/978-3-662-07003-1
18. Nipkow, T., Klein, G.: Concrete semantics: with Isabelle/HOL. Springer (2014). https://doi.org/10.1007/978-3-319-10542-0, http://concrete-semantics.org/
19. Noschinski, L.: A graph library for Isabelle. Mathematics in Computer Science **9**(1), 23–39 (2015). https://doi.org/10.1007/s11786-014-0183-z
20. Paulson, L.C., Nipkow, T., Wenzel, M.: From LCF to Isabelle/HOL. Formal Aspects of Computing **31**(6), 675–698 (2019). https://doi.org/10.1007/s00165-019-00492-1
21. Plump, D.: Reasoning about graph programs. In: Proc. Computing with Terms and Graphs (TERMGRAPH 2016). Electronic Proceedings in Theoretical Computer Science, vol. 225, pp. 35–44 (2016). https://doi.org/10.4204/EPTCS.225.6
22. Rosen, B.K.: Deriving graphs from graphs by applying a production. Acta Informatica **4**, 337–357 (1975)
23. Schirmer, N., Wenzel, M.: State spaces - the locale way. In: Proc. International Workshop on Systems Software Verification (SSV 2009). Electronic Notes in Theoretical Computer Science, vol. 254, pp. 161–179 (2009). https://doi.org/10.1016/j.entcs.2009.09.065

24. Söldner, R., Plump, D.: Towards mechanised proofs in double-pushout graph transformation. In: Proc. International Workshop on Graph Computation Models (GCM 2022). Electronic Proceedings in Theoretical Computer Science, vol. 374, pp. 59–75 (2022). `https://doi.org/10.4204/EPTCS.374.6`
25. Strecker, M.: Interactive and automated proofs for graph transformations. Mathematical Structures in Computer Science **28**(8), 1333–1362 (2018). `https://doi.org/10.1017/S096012951800021X`
26. Wenzel, M.: Isar — A Generic Interpretative Approach to Readable Formal Proof Documents. In: Theorem Proving in Higher Order Logics (TPHOLs 1999). Lecture Notes in Computer Science, vol. 1690, pp. 167–183. Springer (1999). `https://doi.org/10.1007/3-540-48256-3_12`
27. Wulandari, G.S., Plump, D.: Verifying graph programs with monadic second-order logic. In: Proc. International Conference on Graph Transformation (ICGT 2021). Lecture Notes in Computer Science, vol. 12741, pp. 240–261. Springer (2021). `https://doi.org/10.1007/978-3-030-78946-6_13`