CORRECT TRANSFORMATION RULES FOR INCREMENTAL DEVELOPMENT OF ARCHITECTURE HIERARCHIES

R. A. RIEMENSCHNEIDER

ABSTRACT. In previous papers, my colleagues and I defined what it means to say that an architecture refinement hierarchy is *correct*, and showed how to prove correctness of refinement steps. This paper builds upon that earlier work by extending the notion of correctness to apply to architecture transformation rules that can be used to incrementally build hierarchies. Perhaps surprisingly, it is *not* the case that a rule that adds detail to an architectural description at some level of detail is correct only when the addition of detail always produces a correct refinement of the description. There is, nonetheless, a close relationship between proofs that rules are correct and proofs that certain refinement steps are correct. The possibility of defining general techniques for generating correct incremental transformation rules from a correct refinement pattern is explored.

1. INTRODUCTION

In previous papers [3, 5], my colleagues and I defined what it means for a refinement step in an architecture hierarchy[1] to be *correct* and proved the correctness of a refinement pattern that says a dataflow style architectural description consisting of two procedural components connected by a dataflow channel can be implemented as two procedural components and a variable that is written by one procedure and read by the other. This pattern is shown in Figure 1.

However, it should be noted that we did *not* show is that a dataflow channel can *always* be replaced by a shared variable in this way. We proved that refinement step shown in Figure 1 is always correct; we did not prove that the refinement step shown in Figure 2, where the same refinement is embedded in a more complicated context, is always correct. In other words, we did not show that, if an architecture hierarchy is correct, then the extended hierarchy that results by adding a node in which a chosen dataflow channel has been refined into a read and written variable is also correct. We did not show this for a good reason: the extended hierarchy may not be correct. A very simple example illustrates this point.

The two architectural descriptions shown in Figure 3 differ in that more abstract description D_1', which is an extension of description D_1 in Figure 1, contains a dataflow channel c_2 that has been replaced in the more concrete description D_2' by a shared variable v_2. According to our correctness criterion [3], every architectural description in a hierarchy should tell the *whole* truth — more precisely, the whole truth expressible at the level of abstraction determined by the style of the description — about the architecture. Clearly, if description D_2' in Figure 3 is telling the

[1]An architecture hierarchy is a collection of architectural descriptions, at various levels of abstraction and often in a variety of styles, linked by refinement mappings. Since the descriptions are all intended to describe the same software architecture, they must paint a consistent portrait of that architecture. Consistency is guaranteed by the correctness of the refinement steps in the hierarchy. Thus a hierarchy is correct iff every refinement step in it is correct.

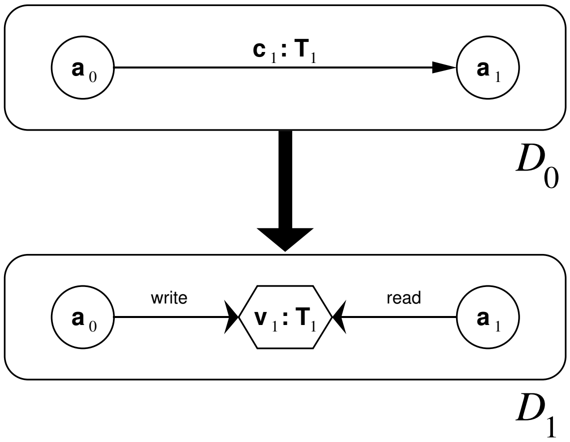

FIGURE 1. Replacing a Dataflow Channel by a Shared Variable

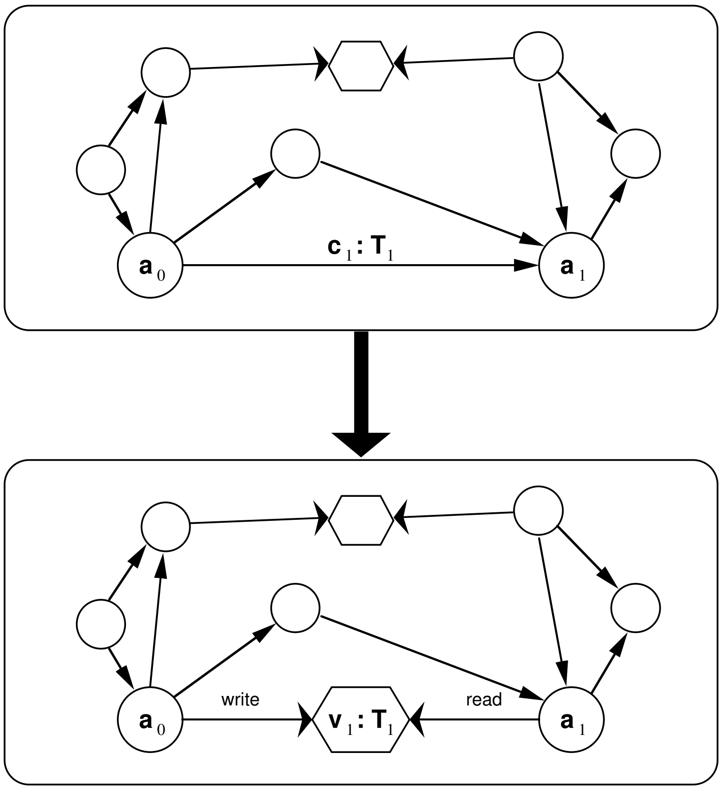

FIGURE 2. Replacing a Dataflow Channel by a Shared Variable, In Context

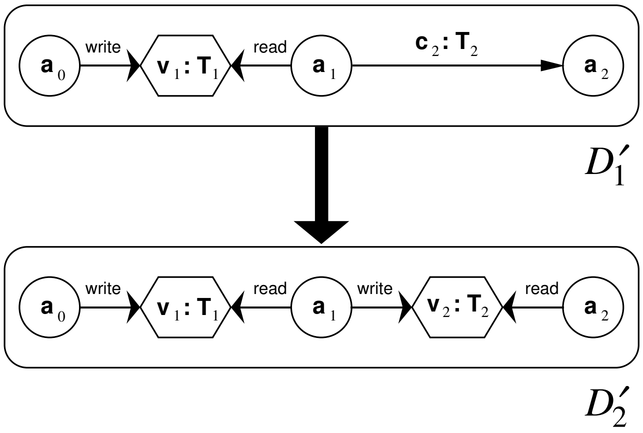

FIGURE 3. An Incorrect Hierarchy

truth, then description D_1' is not telling the whole truth. The language used for D_1' has the descriptive power to say that the dataflow from $\mathbf{a}_0$ to $\mathbf{a}_1$ is implemented as a shared variable: it mentions the shared variable $\mathbf{v}_2$. In formal terms, the natural interpretation of the theory corresponding to the abstract description in the theory corresponding to the concrete description — the one that interprets dataflow in terms of shared variables, and leaves everything else "as is" — is not *faithful*.[2]

2. Composition versus Transformation

2.1. Composition. It is easy to see that the source of the problem is the mixing of dataflow style constructs and shared variable style constructs in D_1'. In our previous paper [3], we employed a development methodology that avoided such non-homogeneous architectural descriptions. The basic idea was that every dataflow channel in a specification would be simultaneously replaced by its implementation. This complex refinement would be proven correct by showing it to be an instance of a "mega-pattern" built from verified primitive refinement patterns, using modes of composition that have been shown to preserve correctness. Figure 4 illustrates a simple case where a refinement of dataflow channel $\mathbf{c}_1$ that is correct in isolation and a refinement of dataflow channel $\mathbf{c}_2$ that is also correct in isolation are composed to obtain a correct refinement of the full dataflow description.

Despite the intuitive appeal of this compositional approach, there are serious difficulties in attempting to characterize the modes of composition that preserve

[2]An *interpretation* $\mathscr{I}$ of theory Θ in theory Υ is a (total) mapping from sentences in the language $\mathscr{L}$ of Θ to sentences in the language of Υ such that, for every $\mathscr{L}$-sentence φ,

$$\varphi \in \Theta \Longrightarrow \mathscr{I}(\varphi) \in \Upsilon$$

If, in addition, for every $\mathscr{L}$-sentence φ,

$$\mathscr{I}(\varphi) \in \Upsilon \Longrightarrow \varphi \in \Theta$$

then interpretation $\mathscr{I}$ is *faithful*. For more detail on interpretations, see my earlier paper [5] on how to prove a mapping is a faithful interpretation.

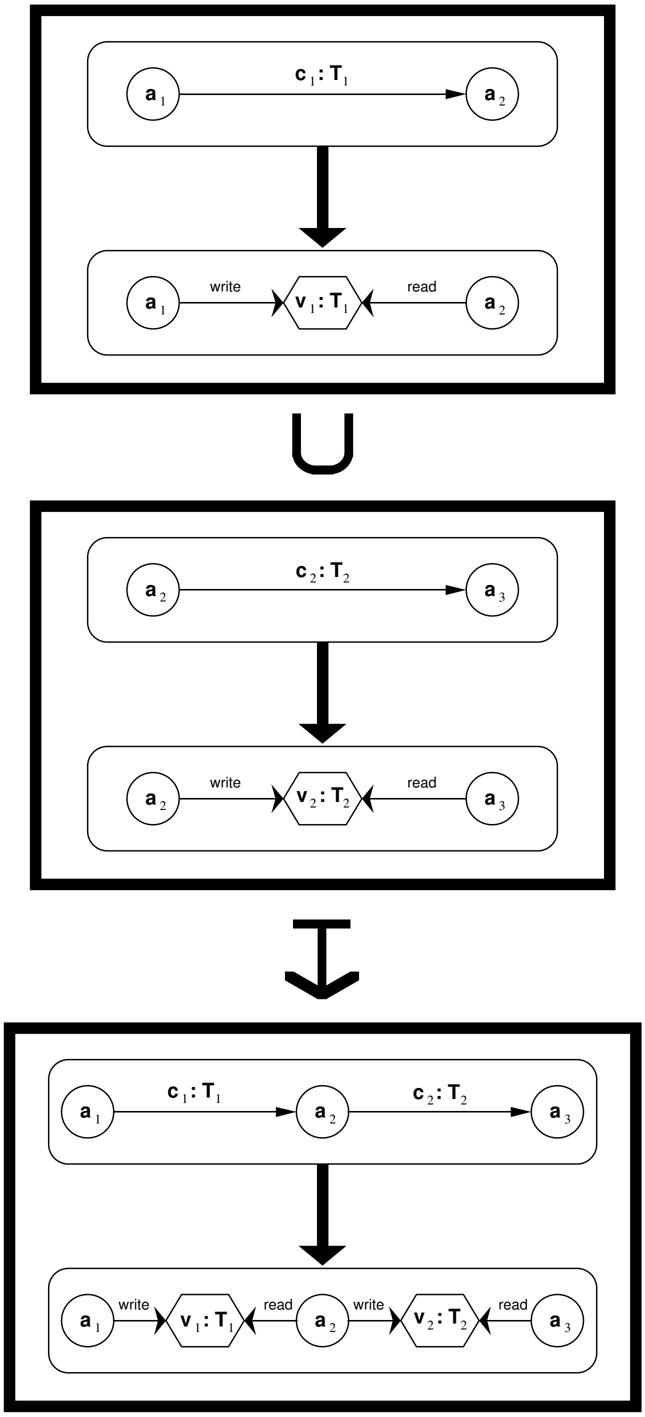

FIGURE 4. Composing Correct Refinements

correctness. It is instructive to see why Moriconi and Qian's attempt to work out the formal details [2, Section 8] is unsatisfactory.

First, their "definition" of $I_1 \cup I_2$ is incomplete. They write

> More precisely, if
>
> $$I_1 : \Theta_1 \to \Theta_1' \quad \text{and} \quad I_2 : \Theta_2 \to \Theta_2'$$
>
> are faithful interpretations, then we want
>
> $$I_1 \cup I_2 : \Theta_1 \cup \Theta_2 \to \Theta_1' \cup \Theta_2'$$
>
> to be a faithful interpretation. (The union of two theories is the deductive closure of the set-theoretic union of the theories.)

But how is the mapping $I_1 \cup I_2$ defined on sentences that are consequences of the union of the two theories that are not in the union of their consequences? And how is it defined on sentences that are not consequences of the union? Getting the answers to these questions right is non-trivial.[3] Without the answers, it is hard to assess the merits of their attempt in detail.

Still, it is clear that there is a fundamental difficulty with their account. What, according to Moriconi and Qian, is wrong with composing the correct refinement of Figure 1 with the trivially correct identity refinement on the rest of the system, to obtain the refinement shown in Figure 3? Simply that their second "general condition" for correct composition

> It must not be possible to infer new facts about the composite abstract architecture from the composite concrete architecture. That is, for language L_1 of Θ_1 and L_2 of Θ_2, if
>
> $$F \text{ is a sentence of } L_1 \cup L_2$$
>
> and
>
> $$\Theta_1' \cup \Theta_2' \vdash I(F)$$
>
> then we must prove that
>
> $$I(\Theta_1) \cup I(\Theta_2) \vdash I(F)$$

is violated. But this condition is just a straightforward restatement of the requirement that the interpretation corresponding to the composite refinement is faithful. They claim that this condition is satisfied in their one specific example, which was extracted from the compiler architecture, *because* the subarchitectures "share no interface points". If they were right, the proof that a particular instance of this mode of composition preserves correctness could be reduced to a simple check that there are no common interface points in the domains of the several refinements being composed. Yet the incorrect refinement step in Figure 3 shows that having no shared interface points is *not* sufficient to guarantee faithfulness of this mode of composition. Moriconi and Qian thus failed to provide a single non-trivial example of a mode of composition that is guaranteed to preserve correctness. If every instance of composition requires a proof that their second "general condition" is satisfied (i.e., requires a proof of faithfulness), the fundamental motivating notion of pre-verifying the refinements and modes of composition has been abandoned.

[3] See Section 4.2.

The difficulties of finding modes of refinement composition that preserve correctness has been noted by other researchers. For example, Broy [1, pp. 3] writes

> Traditionally, compositional notions of specification and refinement are considered hard to obtain. ... Finding compositional specification methods and compositional interaction concepts [for the class of systems dealt with in this paper] is considered a difficult issue.

Of course, Broy is concerned with the usual, more liberal notion of correct refinement that consists of "adding logical properties". His behavior refinements correspond to theory extensions that are not necessarily conservative; his interface and interaction refinements correspond to theory interpretations that are not necessarily faithful. Since the faithfulness requirement we have imposed on on correct structural refinement makes correctness proofs much more difficult, it should not be surprising that finding correctness-reserving modes of structural refinement composition becomes more difficult as well. So the question arises: Is there any alternative to the compositional approach that offers similar benefits?

2.2. Transformation. While the charms of the compositional approach are undeniable, incremental transformation offers an even more attractive alternative. The compositional approach is quite natural if the objective is to verify the correctness of a completed hierarchy. On the incremental approach, the objective is to avoid introducing mistakes during the design process, rather than catching mistakes after they have been made. Architectures are designed incrementally in practice, using an architecture design tool — perhaps a CASE tool, perhaps paper and pencil — to incrementally elaborate a hierarchy. As each design decision is made, the hierarchy is transformed to reflect the current state of the design. If all the transformations of the hierarchy preserve its correctness, and the starting point is a trivially correct refinement-free hierarchy, then the completed hierarchy that eventually results is guaranteed to be correct by construction.

But on the transformational approach, non-homogeneous specifications cannot be avoided. For example, consider an abstract architectural description containing two dataflow channels, such as D'_0 of Figure 5. It should be possible to replace one channel by a shared variable without eliminating the option of later replacing the second by a shared variable as well. At first glance, the series of architectural transformations shown in Figure 5 may seem to cast doubt on our definition of correctness, for there is clearly nothing wrong with this series. What the correctness of this series actually shows is that, whatever it means to say that a transformation rule is *correct*, it cannot be the case that correct transformation is simply a matter of correct refinement. The connection between correct transformation rules and correct refinement patterns must be less direct.

3. From Correct Refinements to Correct Transformations

The first step in establishing the connection between correct refinement and correct transformation is to view transformations as being applied to the entire hierarchy rather than to the individual descriptions in the hierarchy. Figure 6 shows the same two transformations that were shown in Figure 5 from this perspective. Initially, the hierarchy H_0 consists of a single level. This level contains an abstract architectural description D'_0 of system dataflow. The first transformation produces

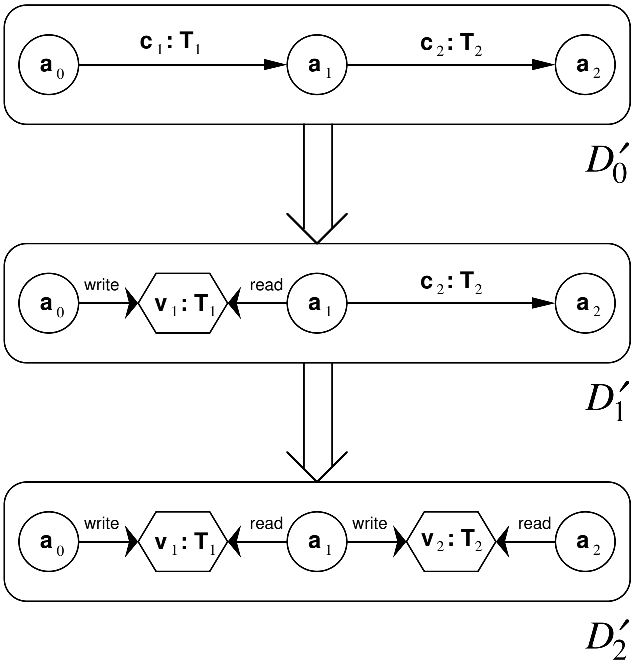

FIGURE 5. Incrementally Replacing Dataflow Channels by Shared Variables

a more elaborate hierarchy H_1 by introducing a second level of description, in which one of the dataflow channels of D'_0 has been replaced by a shared variable. This partly abstract, partly concrete description D'_1 is a correct refinement of D'_0, so H_1 is a correct hierarchy (i.e., every refinement step in the hierarchy is correct). The second transformation, which produces hierarchy H_2, *modifies* the lower level description D'_1 rather than adding another level. The completed concrete level description D'_2 is also a correct refinement of the abstract level description D'_0, making H_2 a correct hierarchy. Note that H_2 is identical to the hierarchy produced by composition in Figure 4.

The original refinement pattern in Figure 1 thus corresponds to two transformation rules, which can be stated informally as follows.

(T_1) If an architectural hierarchy has a maximal node containing a description in pure dataflow style, then a successor of that maximal node can be added containing a description in which one of the dataflow channels of the maximal node's description has been refined into a shared variable.

(T_2) If an architectural hierarchy has a maximal node containing a description including a dataflow channel, and the predecessor of that node contains a description in pure dataflow style,[4] then the maximal node's description can be replaced by a description in which the dataflow channel has been refined into a fresh shared variable.

[4]The rationale for this restriction on the style of the predecessor will become clear when this transformation is proven correct, in Section 4.

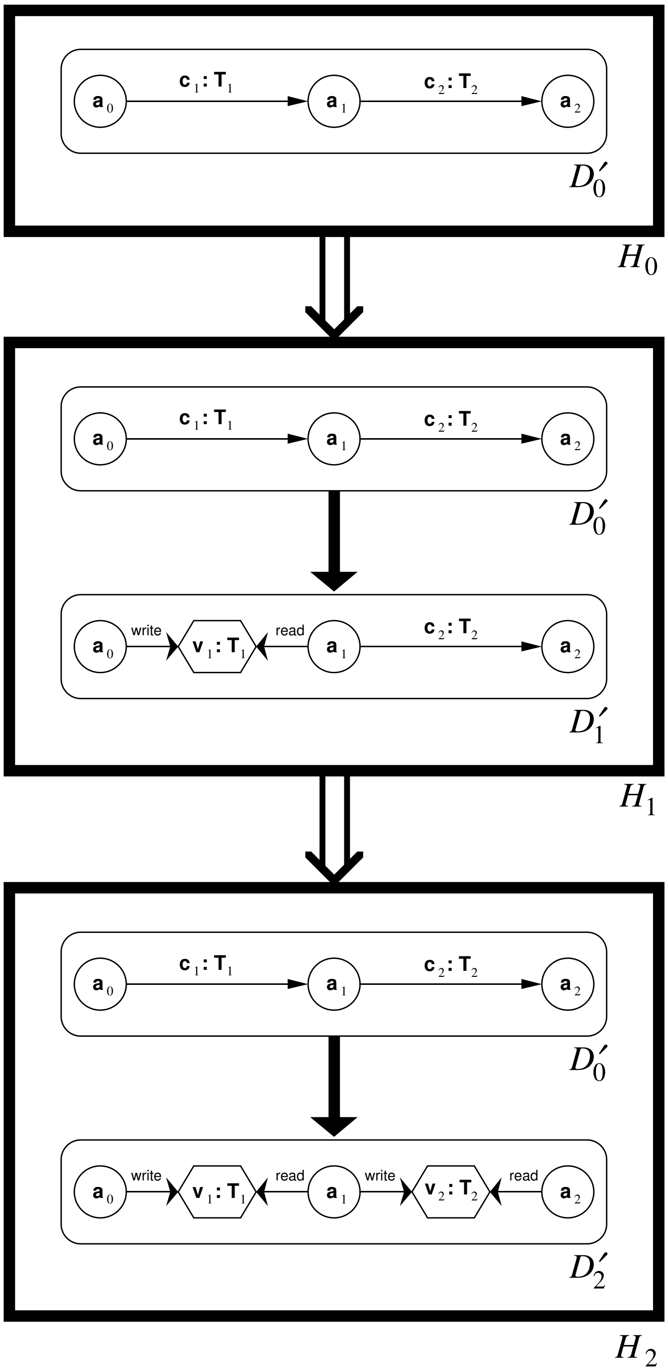

FIGURE 6. Incremental Replacement: Another Perspective

And so the connection between correct refinement and correct transformation is now apparent:

> An architecture hierarchy transformation rule is *correct* if and only if, whenever it is applied to a correct hierarchy (i.e., a hierarchy in which every refinement step is correct), a correct hierarchy results.

In short, correct transformation rules preserve hierarchy correctness.

4. Proving Transformations Correct: Two Approaches

4.1. What more is needed to prove correctness? Transformation T_1, the first of the two transformations above, extends a hierarchy by introducing an additional node, linked by an additional refinement step. So to say that T_1 preserves correctness of hierarchies simply means that the refinement step that is introduced is always correct. Transformation T_2 modifies an existing refinement step, so saying that it is correct means that the modification preserves correctness of the refinement step.

In a previous proof of the correctness of the refinement step in Figure 1 [5, pp. 7–8], the interpretation $\mathscr{K}$ determined by the extending the following basis to every sentence of the language of D_0 in the standard way was shown to be a faithful interpretation of the theory of the more abstract dataflow style architectural description D_0 in the theory of the more concrete shared variable style architectural description D_1.

$$
\begin{aligned}
\omega_{\mathscr{K}} &= \mathsf{x}_0 \equiv \mathsf{x}_0 \\
\mathscr{K}(\mathsf{Procedure}) &= \mathsf{Procedure}(\mathsf{x}_0) \\
\mathscr{K}(\mathsf{Channel}) &= \mathsf{Variable}(\mathsf{x}_0) \\
\mathscr{K}(\mathsf{In\_Port}) &= \mathsf{Call}(\mathsf{x}_0,\mathsf{x}_1) \wedge \exists \mathsf{x}_2\, \mathsf{Can\_Get}(\mathsf{x}_0,\mathsf{x}_2) \\
\mathscr{K}(\mathsf{Out\_Port}) &= \mathsf{Call}(\mathsf{x}_0,\mathsf{x}_1) \wedge \exists \mathsf{x}_2\, \mathsf{Might\_Put}(\mathsf{x}_0,\mathsf{x}_2) \\
\mathscr{K}(\mathsf{Can\_Carry}) &= \mathsf{Can\_Hold}(\mathsf{x}_0,\mathsf{x}_1) \\
\mathscr{K}(\mathsf{Might\_Supply}) &= \mathsf{Might\_Put}(\mathsf{x}_0,\mathsf{x}_1) \\
\mathscr{K}(\mathsf{Can\_Accept}) &= \mathsf{Can\_Get}(\mathsf{x}_0,\mathsf{x}_1) \\
\mathscr{K}(\mathsf{Connects}) &= \mathsf{Writes}(\mathsf{x}_1,\mathsf{x}_0) \wedge \mathsf{Reads}(\mathsf{x}_2,\mathsf{x}_0) \\
\mathscr{K}(\mathbf{T}) &= \mathbf{T}(\mathsf{x}_0) \\
\mathscr{K}(\mathbf{a}_0) &= \mathsf{x}_0 \equiv \mathbf{a}_0 \\
\mathscr{K}(\mathbf{a}_1) &= \mathsf{x}_0 \equiv \mathbf{a}_1 \\
\mathscr{K}(\mathbf{c}_1) &= \mathsf{x}_0 \equiv \mathbf{v}_1 \\
\mathscr{K}(\mathbf{a}_0\mathsf{\_oport}) &= \mathsf{x}_0 \equiv \mathbf{a}_0\mathsf{\_call} \\
\mathscr{K}(\mathbf{a}_1\mathsf{\_iport}) &= \mathsf{x}_0 \equiv \mathbf{a}_1\mathsf{\_call}
\end{aligned}
$$

Why can't the faithfulness of this interpretation be used to prove the refinements introduced by transformation rules T_1 and T_2 are correct? It is not the interpretation that corresponds to these transformations. According to $\mathscr{K}$, *every* channel is implemented as a variable and *every* connection is implemented by reading and writing a shared variable. In incremental development, *some* connecting channels remain connecting channels.

4.2. First approach: modifying the interpretation. One approach to showing T_1 is correct is to modify the definition of interpretation $\mathscr{K}$, in a way that maintains faithfulness, so as to make it correspond to T_1. Ideally, the modification should preserve faithfulness in general, not just in the particular case of $\mathscr{K}$, so that proving a refinement step correct always guarantees that the transformation that extends a hierarchy by applying that refinement to a homogeneous description in a maximal node is also correct.

Unfortunately, well-known general results on extensibility of interpretations are not applicable in this case. For example, Turski and Maibaum's "Modularization Theorem" [6, pp. 174] says that an interpretation of theory Θ in theory Υ can always be extended to an interpretation of any conservative extension Θ' of Θ in a conservative extension of Υ. Despite their claim that this theorem "guarantees that an implementation of [an] extension [of an existing system] will be consistent with an implementation of the existing system," it is not applicable to the sort of "extensions" of descriptions under consideration here. Depending on details of the formalization, the theory of D_0' may or may not be an extension of D_0,[5] but it is certainly not a conservative extension. Sentences made true by the context, such as

$$\exists \mathsf{x}_0\,[\,\mathsf{Procedure}(\mathsf{x}_0) \wedge \neg \mathsf{x}_0 \equiv \mathbf{a}_0 \wedge \neg \mathsf{x}_0 \equiv \mathbf{a}_1\,]$$

will be included in the theory of D_0', but not the theory of D_0. The example Turski and Maibaum use to show that conservativeness is necessary for the theorem to hold also shows that, even if the interpretation to be extended is faithful, conservativeness remains necessary.

Since general results on extensibility do not yield the desired interpretation, a modification suited to our specific requirements must be sought. Changing the definition of $\mathscr{K}$ — thereby defining a new interpretation basis called "$[\mathbf{1} \vee \mathscr{K}]$", for reasons that the definition will make clear — so as to allow the possibility of channels being interpreted as either channels or shared variables is straightforward.

$$\begin{aligned}
\omega_{[\mathbf{1} \vee \mathscr{K}]} &= \mathsf{x}_0 \equiv \mathsf{x}_0 \\
[\mathbf{1} \vee \mathscr{K}]\,(\mathsf{Procedure}) &= \mathsf{Procedure}(\mathsf{x}_0) \\
[\mathbf{1} \vee \mathscr{K}]\,(\mathsf{Channel}) &= \mathsf{Channel}(\mathsf{x}_0) \vee \mathsf{Variable}(\mathsf{x}_0) \\
[\mathbf{1} \vee \mathscr{K}]\,(\mathsf{In_Port}) &= \mathsf{In_Port}(\mathsf{x}_0, \mathsf{x}_1) \\
&\qquad \vee [\,\mathsf{Call}(\mathsf{x}_0, \mathsf{x}_1) \wedge \exists \mathsf{x}_2\, \mathsf{Can_Get}(\mathsf{x}_0, \mathsf{x}_2)\,] \\
[\mathbf{1} \vee \mathscr{K}]\,(\mathsf{Out_Port}) &= \mathsf{Out_Port}(\mathsf{x}_0, \mathsf{x}_1) \\
&\qquad \vee [\,\mathsf{Call}(\mathsf{x}_0, \mathsf{x}_1) \wedge \exists \mathsf{x}_2\, \mathsf{Might_Put}(\mathsf{x}_0, \mathsf{x}_2)\,]
\end{aligned}$$

[5] If the simplified method of proving correctness is used, or the original method is used and "extremal axioms" are added to theories, sentences such as

$$\forall \mathsf{x}_0\,[\,\mathsf{Procedure}(\mathsf{x}_0) \rightarrow \mathsf{x}_0 \equiv \mathbf{a}_0 \vee \mathsf{x}_0 \equiv \mathbf{a}_1\,]$$

will be included in the theory of D_0 but not in the theory of D_0'. As a result, the theory of D_0' will not be an extension of the theory of D_0.

$$
\begin{aligned}
[\mathbf{1} \vee \mathscr{K}]\,(\mathsf{Can\_Carry}) &= \mathsf{Can\_Carry}(\mathsf{x}_0, \mathsf{x}_1) \vee \mathsf{Can\_Hold}(\mathsf{x}_0, \mathsf{x}_1)\\
[\mathbf{1} \vee \mathscr{K}]\,(\mathsf{Might\_Supply}) &= \mathsf{Might\_Supply}(\mathsf{x}_0, \mathsf{x}_1) \vee \mathsf{Might\_Put}(\mathsf{x}_0, \mathsf{x}_1)\\
[\mathbf{1} \vee \mathscr{K}]\,(\mathsf{Can\_Accept}) &= \mathsf{Can\_Accept}(\mathsf{x}_0, \mathsf{x}_1) \vee \mathsf{Can\_Get}(\mathsf{x}_0, \mathsf{x}_1)\\
[\mathbf{1} \vee \mathscr{K}]\,(\mathsf{Connects}) &= \mathsf{Connects}(\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2)\\
&\qquad \vee [\, \mathsf{Writes}(\mathsf{x}_1, \mathsf{x}_0) \wedge \mathsf{Reads}(\mathsf{x}_2, \mathsf{x}_0)\,]\\
[\mathbf{1} \vee \mathscr{K}]\,(\mathbf{T}) &= \mathbf{T}(\mathsf{x}_0)\\
[\mathbf{1} \vee \mathscr{K}]\,(\mathbf{c}_1) &= \mathsf{x}_0 \equiv \mathbf{v}_1\\
[\mathbf{1} \vee \mathscr{K}]\,(\mathbf{a}_0\_\mathsf{oport}) &= \mathsf{x}_0 \equiv \mathbf{a}_0\_\mathsf{call}\\
[\mathbf{1} \vee \mathscr{K}]\,(\mathbf{a}_1\_\mathsf{iport}) &= \mathsf{x}_0 \equiv \mathbf{a}_1\_\mathsf{call}\\
[\mathbf{1} \vee \mathscr{K}]\,(\mathbf{n}) &= \mathsf{x}_0 \equiv \mathbf{n} \qquad \text{for any name } \mathbf{n} \text{ other than}\\
&\qquad\qquad\qquad \mathbf{c}_1,\ \mathbf{a}_0\_\mathsf{oport}, \text{ and } \mathbf{a}_1\_\mathsf{iport}
\end{aligned}
$$

However, the resulting interpretation is not the interpretation that corresponds to transformation T_1, as an example makes clear. Consider the application of T_1 in Figure 6, which produces H_1 from H_0. The intended interpretation of the dataflow-language sentence

$$\mathsf{Channel}(\mathbf{c}_1)$$

is

$$\mathsf{Variable}(\mathbf{v}_1)$$

and the intended interpretation of the sentence

$$\mathsf{Connects}(\mathbf{c}_1, \mathbf{a}_0\_\mathsf{oport}, \mathbf{a}_1\_\mathsf{iport})$$

is

$$\mathsf{Writes}(\mathbf{a}_0\_\mathsf{call}, \mathbf{v}_1) \wedge \mathsf{Reads}(\mathbf{a}_1\_\mathsf{call}, \mathbf{v}_1)$$

According to $[\mathbf{1} \vee \mathscr{K}]$, these two sentences are interpreted as

$$\mathsf{Channel}(\mathbf{v}_1) \vee \mathsf{Variable}(\mathbf{v}_1)$$

and

$$\mathsf{Connects}(\mathbf{c}_1, \mathbf{a}_0\_\mathsf{oport}, \mathbf{a}_1\_\mathsf{iport}) \vee [\, \mathsf{Writes}(\mathbf{a}_0\_\mathsf{call}, \mathbf{v}_1) \wedge \mathsf{Reads}(\mathbf{a}_1\_\mathsf{call}, \mathbf{v}_1)\,]$$

Interpretation $[\mathbf{1} \vee \mathscr{K}]$ gets the general facts right — for example, it correctly interprets "there are at least two dataflow channels" as "the total number of dataflow channels and shared variables that implement dataflow is at least two" — but not the particulars.

Perhaps, then, the interpretation $[\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]]$ that is defined as follows on *atomic* formulas

$$[\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]](\varphi) = \begin{cases} \mathscr{K}(\varphi) & \text{if } \varphi \text{ is a } \textit{sentence} \text{ in the domain of } \mathscr{K}\\ [\mathbf{1} \vee \mathscr{K}](\varphi) & \text{otherwise} \end{cases}$$

and is extended to non-atomic formulas in the standard way corresponds to T_1. (This mode of composition of interpretations — disjunction, but with basic facts treated specially — may be something like what Moriconi and Qian intended for their union operation.) While it is not entirely clear that this is the interpretation that naturally corresponds to T_1, at least there are no obvious counterexamples to that thesis.

The next step is to find the interpretation that corresponds to transformation T_2. The difference here is that T_2 is being used to modify an interpretation $\mathscr{J}$ of a pure dataflow theory in a mixed style theory rather than introduce a new interpretation. But the basic modification being performed, replacement of a dataflow channel by a shared variable, is much the same. If $[\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]]$ is extended so that so that it applies to every sentence in the range of $\mathscr{J}$,

$$[[\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]] \downarrow \mathbf{1}](\varphi) = \begin{cases} [\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]](\varphi) & \text{if } [\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]] \text{ is defined for } \varphi \\ \varphi & \text{otherwise} \end{cases}$$

where the notation "$\downarrow \mathbf{1}$" is intended to suggest "defined everywhere", then T_2 can be thought of as replacing $\mathscr{J}$ by $[[\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]] \downarrow \mathbf{1}] \circ \mathscr{J}$, the mapping that sends the sentence φ to $[[\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]] \downarrow \mathbf{1}]\,(\mathscr{J}(\varphi))$. Again, at least there are no obvious counterexamples to the proposition that $[[\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]] \downarrow \mathbf{1}]$ corresponds to T_2.

At this point, an attempt could be made to prove that $[\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]]$ is a faithful theory interpretation and that, if $\mathscr{J}$ is a faithful theory interpretation, so is $[[\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]] \downarrow \mathbf{1}] \circ \mathscr{J}$. But, regardless of whether the original method or the simplified method of proving faithfulness is employed, it does not seem possible to straightforwardly convert the proof that $\mathscr{K}$ is faithful into the desired proofs.[6]

This analysis suggests that there are two fundamental problems with attempts to use a calculus of interpretations as a basis for proving transformations correct. First, there is the absence of a truly compelling argument that any particular combination of interpretations naturally corresponds to a transformation. No many how many modifications are made, there is no guarantee that yet another example that shows the interpretation fails to correspond to the transformation will not be found. Second, although the correctness of transformations seems to intuitively follow from the correctness of the refinements that suggest them, there does not seem to be any simple, uniform method of converting refinement correctness proofs into transformation correctness proofs. The examples that inspired the modifications show that the interpretations that correspond to these transformations must not be standard — the interpretation of predicates Channel, Connects, and so on, depends on their arguments — and so neither of the techniques we have developed for proving faithfulness can be directly applied. It is not clear how either the "inverse mapping" of the original method or the abstraction mapping on models of the simplified method can be defined. In the next section, an alternative approach to proving transformation correctness that avoids both these problems will be investigated.[7]

[6] When using the original method, architectural descriptions are treated as collections of axioms, mappings are proved to be interpretations by proving the images of the axioms of the abstract theory from the axioms of the concrete theory, and faithfulness if proved by a model theoretic method that requires "inverting" the interpretation. See our earlier paper [3] for details. When using the simplified method, architectural descriptions are taken as specifying a particular structure, mappings are proved to be theory interpretations by using the fact that a mapping of the theory of the abstract structure to the theory of the concrete structure is an interpretation if the image of the concrete structure under the abstraction map that corresponds to the mapping on sentences is isomorphic to the abstract structure, and faithfulness is automatic. See my earlier paper [5] for details.

[7] It will turn out that $[\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]]$ does not correspond to T_1, and that $[[\mathscr{K}; [\mathbf{1} \vee \mathscr{K}]] \downarrow \mathbf{1}]$ does

4.3. Second approach: modifying the theories. The difficulties in adapting $\mathscr{K}$ to generally interpret dataflow in terms of shared variables flow from a single source: the logical theories being used to describe the architectures are too weak to distinguish between dataflow channels that get implemented as shared variables and those that do not. In terms of the example in Figure 6, the interpretation that T_1 introduces — call it $\mathscr{K}^+$ — should act as follows:

$$\mathscr{K}^+(\mathsf{Channel}(\mathbf{c}_1)) = \mathsf{Variable}(\mathbf{v}_1)$$
$$\mathscr{K}^+(\mathsf{Channel}(\mathbf{c}_2)) = \mathsf{Channel}(\mathbf{c}_2)$$

Since these sentences are being interpreted differently by $\mathscr{K}^+$, there is clearly some difference between the two cases that is not being reflected in the logical theory. Rather than making somewhat ad hoc modifications to $\mathscr{K}$ to account for the difference, as in the first approach, let us explore the option of modifying the logical theories.

As a first step, expand the language of dataflow $\mathscr{L}$ used to describe the more abstract architecture. Corresponding to the singulary predicate Channel, add two new singulary predicates Green_Channel and Red_Channel, and similarly for all the other predicates — In_Port, Out_Port, Can_Carry, Might_Supply, Can_Accept, and Connects — that are re-interpreted by $\mathscr{K}$. Call this expanded language $\mathscr{L}^+$.

Next, if the original correctness method is being used, extend the theory Θ of the dataflow description by adding sentences that describe the particular coloring,

$$\begin{array}{l}\mathsf{Green\_Channel}(\mathbf{c})\\ \mathsf{Green\_Connects}(\mathbf{c}, \mathbf{p}_{\mathrm{out}}, \mathbf{p}_{\mathrm{in}})\\ \mathsf{Green\_In\_Port}(\mathbf{p}_{\mathrm{in}}, \mathbf{a}_{\mathrm{in}})\\ \mathsf{Green\_Out\_Port}(\mathbf{p}_{\mathrm{out}}, \mathbf{a}_{\mathrm{out}})\\ \vdots\end{array}$$

where $\mathbf{c}$ is the channel that gets implemented by the application of T_1, $\mathbf{p}_{\mathrm{in}}$ and $\mathbf{p}_{\mathrm{out}}$ are the input and output ports that $\mathbf{c}$ connects, $\mathbf{a}_{\mathrm{in}}$ is the process $\mathbf{p}_{\mathrm{in}}$ is a port of, etc., and

$$\begin{array}{l}\mathsf{Red\_Channel}(\mathbf{c})\\ \mathsf{Red\_Connects}(\mathbf{c}, \mathbf{p}_{\mathrm{out}}, \mathbf{p}_{\mathrm{in}})\\ \mathsf{Red\_In\_Port}(\mathbf{p}_{\mathrm{in}}, \mathbf{a}_{\mathrm{in}})\\ \mathsf{Red\_Out\_Port}(\mathbf{p}_{\mathrm{out}}, \mathbf{a}_{\mathrm{out}})\\ \vdots\end{array}$$

for all other channels $\mathbf{c}$. The extension also adds general *color constraints* that relate colored and non-colored predicates, sentences such as

$$\forall \mathsf{x}_0\,[\,\mathsf{Green\_Channel}(\mathsf{x}_0) \rightarrow \mathsf{Channel}(\mathsf{x}_0)\,]$$

and

$$\forall \mathsf{x}_0\,[\,\mathsf{Red\_Channel}(\mathsf{x}_0) \rightarrow \mathsf{Channel}(\mathsf{x}_0)\,]$$

---

not correspond to T_2. The interpretations that corresponds to these transformations do not even respect logic. So it is doubtful whether additional iterations of "find a counterexample, then modify the definition to eliminate it" would result in discovering correct definitions.

and also sentences such as

$$\begin{aligned}\forall \mathsf{x}_0\, \forall \mathsf{x}_1\, \forall \mathsf{x}_2\, [\, &\mathsf{Green\_Connects}(\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2) \\ &\leftrightarrow \mathsf{Connects}(\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2) \wedge \mathsf{Green\_Channel}(\mathsf{x}_0) \\ &\qquad \wedge \exists \mathsf{x}_3\, \mathsf{Green\_Out\_Port}(\mathsf{x}_1, \mathsf{x}_3) \wedge \exists \mathsf{x}_3\, \mathsf{Green\_In\_Port}(\mathsf{x}_2, \mathsf{x}_3)\,]\end{aligned}$$

and

$$\begin{aligned}\forall \mathsf{x}_0\, \forall \mathsf{x}_1\, \forall \mathsf{x}_2\, [\, &\mathsf{Red\_Connects}(\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2) \\ &\leftrightarrow \mathsf{Connects}(\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2) \wedge \mathsf{Red\_Channel}(\mathsf{x}_0) \\ &\qquad \wedge \exists \mathsf{x}_3\, \mathsf{Red\_Out\_Port}(\mathsf{x}_1, \mathsf{x}_3) \wedge \exists \mathsf{x}_3\, \mathsf{Red\_In\_Port}(\mathsf{x}_2, \mathsf{x}_3)\,]\end{aligned}$$

Call this extension Θ^+. Note that Θ^+ is clearly a conservative extension of Θ. A model $\mathfrak{A}$ of Θ can be expanded to a model $\mathfrak{A}^+$ of Θ^+ simply by assigning the appropriate extensions to the new predicates. For example,

$$\mathsf{Green\_Channel}^{\mathfrak{A}^+} = \{\mathbf{c}^{\mathfrak{A}^+}\}$$

and

$$\mathsf{Red\_Channel}^{\mathfrak{A}^+} = \mathsf{Channel}^{\mathfrak{A}^+} \sim \{\mathbf{c}^{\mathfrak{A}^+}\}$$

where $\mathbf{c}$ is the channel that is implemented by T_1.

Alternatively, if the simplified method is being used, just expand $\mathfrak{A}$ to $\mathfrak{A}^+$ as above, and let Θ^+ be the theory of $\mathfrak{A}^+$. (Note that this Θ^+ contains each of the sentences that was added to Θ when using the original method.) In this case too, Θ^+ is clearly a conservative extension of Θ.

Call this process of adding either $\mathsf{Green\_}$ or $\mathsf{Red\_}$ to predicates of atomic sentences to indicate how their arguments are being acted upon by the transformation *coloring*. For any $\mathscr{L}$-sentence φ, let φ^+, the *coloring of* φ, be the $\mathscr{L}^+$-sentence

$$\varphi \wedge \bigwedge_{\mathbf{n} \in \mathbf{N}} \tau_{\mathbf{n}}$$

where $\mathbf{N}$ is the set of names that occur in φ and, for any name $\mathbf{n}$, the sentence $\tau_{\mathbf{n}}$ gives the most explicit typing of the denotation of $\mathbf{n}$. For example, $\tau_{\mathbf{c}}$ is $\mathsf{Green\_Channel}(\mathbf{c})$ if $\mathbf{c}$ is the name of the channel being implemented by the application of T_1, and $\tau_{\mathbf{c}}$ is $\mathsf{Red\_Channel}(\mathbf{c})$ if $\mathbf{c}$ is the name of any other channel.

Observation 1. Given the way Θ^+ has been constructed from Θ, regardless of choice of method, it is easy to see that, for every $\mathscr{L}$-sentence φ,

$$\varphi \in \Theta \iff \varphi^+ \in \Theta^+$$

Defining a standard interpretation $\mathscr{K}_+$ of Θ^+ in the language of mixed dataflow and shared variables is now straightforward. Let $\mathbf{c}$ be the name of the channel being implemented, $\mathbf{v}$ be the name of the implementing variable, $\mathbf{p}_{\mathrm{out}}$ be the name of the output port being implemented, $\mathbf{k}_{\mathrm{write}}$ be the name of the implementing write call,

$\mathbf{p}_{\mathrm{in}}$ be the name of the input port being implemented, and $\mathbf{k}_{\mathrm{read}}$ be the name of the implementing read call.

$$
\begin{aligned}
\omega_{\mathscr{K}_+} &= \mathsf{x}_0 \equiv \mathsf{x}_0 \\
\mathscr{K}_+\,(\mathsf{Procedure}) &= \mathsf{Procedure}(\mathsf{x}_0) \\
\mathscr{K}_+\,(\mathsf{Channel}) &= \mathsf{Channel}(\mathsf{x}_0) \vee \mathsf{Variable}(\mathsf{x}_0) \\
\mathscr{K}_+\,(\mathsf{Green\_Channel}) &= \mathsf{Variable}(\mathsf{x}_0) \\
\mathscr{K}_+\,(\mathsf{Red\_Channel}) &= \mathsf{Channel}(\mathsf{x}_0) \\
\mathscr{K}_+\,(\mathsf{In\_Port}) &= \mathsf{In\_Port}(\mathsf{x}_0, \mathsf{x}_1) \\
&\qquad \vee [\, \mathsf{Call}(\mathsf{x}_0, \mathsf{x}_1) \wedge \exists \mathsf{x}_2\, \mathsf{Can\_Get}(\mathsf{x}_0, \mathsf{x}_2) \,] \\
\mathscr{K}_+\,(\mathsf{Green\_In\_Port}) &= \mathsf{Call}(\mathsf{x}_0, \mathsf{x}_1) \wedge \exists \mathsf{x}_2\, \mathsf{Can\_Get}(\mathsf{x}_0, \mathsf{x}_2) \\
\mathscr{K}_+\,(\mathsf{Red\_In\_Port}) &= \mathsf{In\_Port}(\mathsf{x}_0, \mathsf{x}_1) \\
\mathscr{K}_+\,(\mathsf{Out\_Port}) &= \mathsf{Out\_Port}(\mathsf{x}_0, \mathsf{x}_1) \\
&\qquad \vee [\, \mathsf{Call}(\mathsf{x}_0, \mathsf{x}_1) \wedge \exists \mathsf{x}_2\, \mathsf{Might\_Put}(\mathsf{x}_0, \mathsf{x}_2) \,] \\
\mathscr{K}_+\,(\mathsf{Green\_Out\_Port}) &= \mathsf{Call}(\mathsf{x}_0, \mathsf{x}_1) \wedge \exists \mathsf{x}_2\, \mathsf{Might\_Put}(\mathsf{x}_0, \mathsf{x}_2) \\
\mathscr{K}_+\,(\mathsf{Red\_Out\_Port}) &= \mathsf{Out\_Port}(\mathsf{x}_0, \mathsf{x}_1) \\
\mathscr{K}_+\,(\mathsf{Can\_Carry}) &= \mathsf{Can\_Carry}(\mathsf{x}_0, \mathsf{x}_1) \vee \mathsf{Can\_Hold}(\mathsf{x}_0, \mathsf{x}_1) \\
\mathscr{K}_+\,(\mathsf{Green\_Can\_Carry}) &= \mathsf{Can\_Hold}(\mathsf{x}_0, \mathsf{x}_1) \\
\mathscr{K}_+\,(\mathsf{Red\_Can\_Carry}) &= \mathsf{Can\_Carry}(\mathsf{x}_0, \mathsf{x}_1) \\
\mathscr{K}_+\,(\mathsf{Might\_Supply}) &= \mathsf{Might\_Supply}(\mathsf{x}_0, \mathsf{x}_1) \vee \mathsf{Might\_Put}(\mathsf{x}_0, \mathsf{x}_1) \\
\mathscr{K}_+\,(\mathsf{Green\_Might\_Supply}) &= \mathsf{Might\_Put}(\mathsf{x}_0, \mathsf{x}_1) \\
\mathscr{K}_+\,(\mathsf{Red\_Might\_Supply}) &= \mathsf{Might\_Supply}(\mathsf{x}_0, \mathsf{x}_1) \\
\mathscr{K}_+\,(\mathsf{Can\_Accept}) &= \mathsf{Can\_Accept}(\mathsf{x}_0, \mathsf{x}_1) \vee \mathsf{Can\_Get}(\mathsf{x}_0, \mathsf{x}_1) \\
\mathscr{K}_+\,(\mathsf{Green\_Can\_Accept}) &= \mathsf{Can\_Get}(\mathsf{x}_0, \mathsf{x}_1) \\
\mathscr{K}_+\,(\mathsf{Red\_Can\_Accept}) &= \mathsf{Can\_Accept}(\mathsf{x}_0, \mathsf{x}_1) \\
\mathscr{K}_+\,(\mathsf{Connects}) &= \mathsf{Connects}(\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2) \\
&\qquad \vee [\, \mathsf{Writes}(\mathsf{x}_1, \mathsf{x}_0) \wedge \mathsf{Reads}(\mathsf{x}_2, \mathsf{x}_0) \,] \\
\mathscr{K}_+\,(\mathsf{Green\_Connects}) &= \mathsf{Writes}(\mathsf{x}_1, \mathsf{x}_0) \wedge \mathsf{Reads}(\mathsf{x}_2, \mathsf{x}_0) \\
\mathscr{K}_+\,(\mathsf{Red\_Connects}) &= \mathsf{Connects}(\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2) \\
\mathscr{K}_+\,(\mathbf{T}) &= \mathbf{T}(\mathsf{x}_0) \\
\mathscr{K}_+\,(\mathbf{c}) &= \mathsf{x}_0 \equiv \mathbf{v} \\
\mathscr{K}_+\,(\mathbf{p}_{\mathrm{out}}) &= \mathsf{x}_0 \equiv \mathbf{k}_{\mathrm{write}} \\
\mathscr{K}_+\,(\mathbf{p}_{\mathrm{in}}) &= \mathsf{x}_0 \equiv \mathbf{k}_{\mathrm{read}} \\
\mathscr{K}_+\,(\mathbf{n}) &= \mathsf{x}_0 \equiv \mathbf{n} \qquad \text{for any name } \mathbf{n} \text{ other than} \\
&\qquad\qquad\quad\; \mathbf{c},\ \mathbf{p}_{\mathrm{out}}, \text{ and } \mathbf{p}_{\mathrm{out}}
\end{aligned}
$$

Interpretation $\mathscr{K}^+$ can now be defined, for every $\mathscr{L}$-sentence φ, by

$$\mathscr{K}^+(\varphi) = \mathscr{K}_+(\varphi^+)$$

It should be clear that $\mathscr{K}^+$, as defined above, is the interpretation that corresponds to T_1, but consideration of an example will reinforce this point. Consider the dataflow language sentence

$$\mathsf{Connects}(\mathbf{c}, \mathbf{p}_{\text{out}}, \mathbf{p}_{\text{in}})$$

where $\mathbf{c}$ denotes the channel being implemented by T_1, and $\mathbf{p}_{\text{out}}$ and $\mathbf{p}_{\text{in}}$ denote the ports that $\mathbf{c}$ connects. The result of coloring this sentence is

$$\begin{aligned}&\mathsf{Connects}(\mathbf{c}, \mathbf{p}_{\text{out}}, \mathbf{p}_{\text{in}}) \wedge \mathsf{Green\_Channel}(\mathbf{c})\\ &\qquad \wedge \exists \mathsf{x}_0\, \mathsf{Green\_Out\_Port}(\mathbf{p}_{\text{out}}, \mathsf{x}_0) \wedge \exists \mathsf{x}_0\, \mathsf{Green\_In\_Port}(\mathbf{p}_{\text{in}}, \mathsf{x}_0)\end{aligned}$$

Note that this sentence is equivalent, modulo the color and type constraints of Θ^+, to

$$\mathsf{Green\_Connects}(\mathbf{c}, \mathbf{p}_{\text{out}}, \mathbf{p}_{\text{in}})$$

Applying $\mathscr{K}_+$ to the colored sentence results in the sentence

$$\begin{aligned}&[\,\mathsf{Connects}(\mathbf{v}, \mathbf{k}_{\text{write}}, \mathbf{k}_{\text{read}}) \vee [\,\mathsf{Writes}(\mathbf{k}_{\text{write}}, \mathbf{v}) \wedge \mathsf{Reads}(\mathbf{k}_{\text{read}}, \mathbf{v})\,]]\\ &\qquad\qquad \wedge \mathsf{Variable}(\mathbf{v})\\ &\qquad\qquad \wedge \exists \mathsf{x}_0\,[\,\mathsf{Call}(\mathbf{k}_{\text{write}}, \mathsf{x}_0) \wedge \exists \mathsf{x}_1\, \mathsf{Might\_Put}(\mathbf{k}_{\text{write}}, \mathsf{x}_0)]\\ &\qquad\qquad \wedge \exists \mathsf{x}_0\,[\,\mathsf{Call}(\mathbf{k}_{\text{read}}, \mathsf{x}_0) \wedge \exists \mathsf{x}_1\, \mathsf{Can\_Get}(\mathbf{k}_{\text{read}}, \mathsf{x}_0)]\end{aligned}$$

where $\mathbf{v}$ denotes the implementing variable, and $\mathbf{k}_{\text{write}}$ and $\mathbf{k}_{\text{read}}$ denote the implementing calls. This sentence is equivalent, modulo the type constraints of Υ, to

$$\mathsf{Writes}(\mathbf{k}_{\text{write}}, \mathbf{v}) \wedge \mathsf{Reads}(\mathbf{k}_{\text{read}}, \mathbf{v})$$

as desired.

4.4. Proving $\mathscr{K}_+$ and $\mathscr{K}^+$ faithful. Interpretation $\mathscr{K}_+$ can be shown to be a faithful interpretation of Θ^+ in the theory Υ of the mixed architecture that is introduced by the application of T_1 using essentially the same proof used to show that $\mathscr{K}$ is a faithful interpretation of the theory of the piece of the dataflow architecture that is being implemented in the theory of the implementing piece of the shared variable architecture. A sketch of the proof that $\mathscr{K}_+$ faithfully interprets the colored theory of D'_0 in the theory of D'_1 illustrates this point.

The structures $\mathfrak{D}'_0$ that correspond to D'_0 can be formally defined as follows:

$$\begin{aligned}|\mathfrak{D}'_0| &= \{a_0, a_1, a_2, c_1, c_2, p_0, p_1, p_2, p_3\} \cup T_1 \cup T_2\\ \mathsf{Procedure}^{\mathfrak{D}'_0} &= \{a_0, a_1, a_2\}\\ \mathsf{Channel}^{\mathfrak{D}'_0} &= \{c_1, c_2\}\\ \mathsf{Green\_Channel}^{\mathfrak{D}'_0} &= \{c_1\}\\ \mathsf{Red\_Channel}^{\mathfrak{D}'_0} &= \{c_2\}\\ \mathsf{In\_Port}^{\mathfrak{D}'_0} &= \{\langle p_1, a_1\rangle, \langle p_3, a_2\rangle\}\end{aligned}$$

$$
\begin{aligned}
\mathsf{Green\_In\_Port}^{\mathfrak{D}'_0} &= \{\langle p_1, a_1\rangle\}\\
\mathsf{Red\_In\_Port}^{\mathfrak{D}'_0} &= \{\langle p_3, a_2\rangle\}\\
\mathsf{Out\_Port}^{\mathfrak{D}'_0} &= \{\langle p_0, a_0\rangle, \langle p_2, a_1\rangle\}\\
\mathsf{Green\_Out\_Port}^{\mathfrak{D}'_0} &= \{\langle p_0, a_0\rangle\}\\
\mathsf{Red\_Out\_Port}^{\mathfrak{D}'_0} &= \{\langle p_2, a_1\rangle\}\\
\mathsf{Can\_Carry}^{\mathfrak{D}'_0} &= \{\langle c_1, t\rangle : t \in T_1\} \cup \{\langle c_2, t\rangle : t \in T_2\}\\
\mathsf{Green\_Can\_Carry}^{\mathfrak{D}'_0} &= \{\langle c_1, t\rangle : t \in T_1\}\\
\mathsf{Red\_Can\_Carry}^{\mathfrak{D}'_0} &= \{\langle c_2, t\rangle : t \in T_2\}\\
\mathsf{Might\_Supply}^{\mathfrak{D}'_0} &= \{\langle p_0, t\rangle : t \in T_1\} \cup \{\langle p_2, t\rangle : t \in T_2\}\\
\mathsf{Green\_Might\_Supply}^{\mathfrak{D}'_0} &= \{\langle p_0, t\rangle : t \in T_1\}\\
\mathsf{Red\_Might\_Supply}^{\mathfrak{D}'_0} &= \{\langle p_2, t\rangle : t \in T_2\}\\
\mathsf{Can\_Accept}^{\mathfrak{D}'_0} &= \{\langle p_1, t\rangle : t \in T_1\} \cup \{\langle p_3, t\rangle : t \in T_2\}\\
\mathsf{Green\_Can\_Accept}^{\mathfrak{D}'_0} &= \{\langle p_1, t\rangle : t \in T_1\}\\
\mathsf{Red\_Can\_Accept}^{\mathfrak{D}'_0} &= \{\langle p_3, t\rangle : t \in T_2\}\\
\mathsf{Connects}^{\mathfrak{D}'_0} &= \{\langle c_1, p_0, p_1\rangle, \langle c_2, p_2, p_3\rangle\}\\
\mathsf{Green\_Connects}^{\mathfrak{D}'_0} &= \{\langle c_1, p_0, p_1\rangle\}\\
\mathsf{Red\_Connects}^{\mathfrak{D}'_0} &= \{\langle c_2, p_2, p_3\rangle\}\\
\mathbf{T}_1^{\mathfrak{D}'_0} &= T_1\\
\mathbf{T}_2^{\mathfrak{D}'_0} &= T_2\\
\mathbf{a}_0^{\mathfrak{D}'_0} &= a_0\\
\mathbf{a}_1^{\mathfrak{D}'_0} &= a_1\\
\mathbf{a}_2^{\mathfrak{D}'_0} &= a_2\\
\mathbf{c}_1^{\mathfrak{D}'_0} &= c_1\\
\mathbf{c}_2^{\mathfrak{D}'_0} &= c_2\\
\mathbf{a}_0\mathsf{\_oport}^{\mathfrak{D}'_0} &= p_0\\
\mathbf{a}_1\mathsf{\_iport}^{\mathfrak{D}'_0} &= p_1\\
\mathbf{a}_1\mathsf{\_oport}^{\mathfrak{D}'_0} &= p_2\\
\mathbf{a}_2\mathsf{\_iport}^{\mathfrak{D}'_0} &= p_3
\end{aligned}
$$

where a_0, a_1, a_2, c_1, c_2, p_0, p_1, p_2, and p_3 are some nine distinct objects, none of which is a member of the set $T_1 \cup T_2$. Similarly, the structures $\mathfrak{D}'_1$ that correspond to D_1' can be formally defined as follows:

$$\begin{aligned}
|\mathfrak{D}_1'| &= \{a_0, a_1, a_2, v_1, c_2, k_0, k_1, p_2, p_3\} \cup T_1 \cup T_2 \\
\mathsf{Procedure}^{\mathfrak{D}'_1} &= \{a_0, a_1, a_2\} \\
\mathsf{Variable}^{\mathfrak{D}'_1} &= \{v_1\} \\
\mathsf{Channel}^{\mathfrak{D}'_1} &= \{c_2\} \\
\mathsf{Call}^{\mathfrak{D}'_1} &= \{\langle k_0, a_0\rangle, \langle k_1, a_1\rangle\} \\
\mathsf{In_Port}^{\mathfrak{D}'_1} &= \{\langle p_3, a_2\rangle\} \\
\mathsf{Out_Port}^{\mathfrak{D}'_1} &= \{\langle p_2, a_1\rangle\} \\
\mathsf{Can_Hold}^{\mathfrak{D}'_1} &= \{\langle v_1, t\rangle : t \in T_1\} \\
\mathsf{Can_Carry}^{\mathfrak{D}'_1} &= \{\langle c_2, t\rangle : t \in T_2\} \\
\mathsf{Might_Put}^{\mathfrak{D}'_1} &= \{\langle k_0, t\rangle : t \in T_1\} \\
\mathsf{Might_Supply}^{\mathfrak{D}'_1} &= \{\langle p_2, t\rangle : t \in T_2\} \\
\mathsf{Can_Get}^{\mathfrak{D}'_1} &= \{\langle k_1, t\rangle : t \in T_1\} \\
\mathsf{Can_Accept}^{\mathfrak{D}'_1} &= \{\langle p_3, t\rangle : t \in T_2\} \\
\mathsf{Writes}^{\mathfrak{D}'_1} &= \{\langle k_0, v_1\rangle\} \\
\mathsf{Reads}^{\mathfrak{D}'_1} &= \{\langle k_1, v_1\rangle\} \\
\mathbf{T}_1^{\mathfrak{D}'_1} &= T_1 \\
\mathbf{T}_2^{\mathfrak{D}'_1} &= T_2 \\
\mathbf{a}_0^{\mathfrak{D}'_1} &= a_0 \\
\mathbf{a}_1^{\mathfrak{D}'_1} &= a_1 \\
\mathbf{a}_2^{\mathfrak{D}'_1} &= a_2 \\
\mathbf{v}_1^{\mathfrak{D}'_1} &= v_1 \\
\mathbf{c}_2^{\mathfrak{D}'_1} &= c_2 \\
\mathbf{a}_0\mathsf{_call}^{\mathfrak{D}'_1} &= k_0 \\
\mathbf{a}_1\mathsf{_call}^{\mathfrak{D}'_1} &= k_1 \\
\mathbf{a}_1\mathsf{_oport}^{\mathfrak{D}'_0} &= p_2 \\
\mathbf{a}_2\mathsf{_iport}^{\mathfrak{D}'_0} &= p_3
\end{aligned}$$

where a_0, a_1, a_2, v_1, c_2, k_0, k_1, p_2, and p_3 are some nine distinct objects, none of which is a member of the set $T_1 \cup T_2$.

The calculation of the value of the abstraction map $\mathscr{K}_+^\partial$ at $\mathfrak{D}_1'$ is very much like the calculation of $\mathscr{K}^{\partial}$ at $\mathfrak{D}_1$: the $\mathsf{Green\_}$ predicates correspond to the predicates of the language of D_0, the $\mathsf{Red\_}$ predicates are the identity, and the non-colored predicates are simply unions. I will show the calculation for just one triple of predicates, since they are all so similar.

$$\begin{aligned}
\mathsf{Channel}^{\mathscr{K}_+^{\partial}(\mathfrak{D}'_1)} &= \{x_0 \in |\mathfrak{D}'_1| : \mathfrak{D}'_1 \models \mathscr{K}_+(\mathsf{Channel})\,[x_0]\} \\
&= \{x_0 \in |\mathfrak{D}'_1| : \mathfrak{D}'_1 \models \mathsf{Channel(x_0)} \vee \mathsf{Variable(x_0)}\,[x_0]\} \\
&= \{v_1, c_2\}
\end{aligned}$$

$$\begin{aligned}
\mathsf{Green_Channel}^{\mathscr{K}_+^{\partial}(\mathfrak{D}'_1)} &= \{x_0 \in |\mathfrak{D}'_1| : \mathfrak{D}'_1 \models \mathscr{K}_+(\mathsf{Channel})\,[x_0]\} \\
&= \{x_0 \in |\mathfrak{D}'_1| : \mathfrak{D}'_1 \models \mathsf{Variable(x_0)}\,[x_0]\} \\
&= \{v_1\}
\end{aligned}$$

$$\begin{aligned}
\mathsf{Red_Channel}^{\mathscr{K}_+^{\partial}(\mathfrak{D}'_1)} &= \{x_0 \in |\mathfrak{D}'_1| : \mathfrak{D}'_1 \models \mathscr{K}_+(\mathsf{Channel})\,[x_0]\} \\
&= \{x_0 \in |\mathfrak{D}'_1| : \mathfrak{D}'_1 \models \mathsf{Channel(x_0)}\,[x_0]\} \\
&= \{c_2\}
\end{aligned}$$

And, just as in the proof of $\mathscr{K}$, it is easy to see that the function h from $|\mathfrak{D}'_0|$ to $|\mathscr{K}_+^{\partial}(\mathfrak{D}'_1)|$ defined by

$$\begin{aligned}
h(a_0) &= a_0 \\
h(a_1) &= a_1 \\
h(a_2) &= a_2 \\
h(c_1) &= v_1 \\
h(c_2) &= c_2 \\
h(p_0) &= k_0 \\
h(p_1) &= k_1 \\
h(p_2) &= p_2 \\
h(p_3) &= p_3 \\
h(t) &= t \qquad \text{for every } t \in T_1 \cup T_2
\end{aligned}$$

is an isomorphism. Hence, $\mathscr{K}_+$ is faithful.

That $\mathscr{K}^+$ is a faithful interpretation of Θ in Υ is an immediate corollary: for every $\mathscr{L}$-sentence φ,

$$\begin{aligned}
\varphi \in \Theta \quad &\iff \quad \varphi^+ \in \Theta^+ && \text{(Observation 1)} \\
&\iff \quad \mathscr{K}_+(\varphi^+) \in \Upsilon && (\mathscr{K}_+ \text{ is faithful}) \\
&\iff \quad \mathscr{K}^+(\varphi) \in \Upsilon && (\text{Definition of } \mathscr{K}^+)
\end{aligned}$$

This completes the proof that T_1 is correct. In contrast to the earlier attempt to modify $\mathscr{K}$ to deal with examples correctly, there can be little doubt that $\mathscr{K}^+$ corresponds to T_1: it was defined to be the result of applying $\mathscr{K}$ to a designated piece of the dataflow architecture — the piece that was "colored green" — and leaving the rest alone. In addition, note that the bulk of the correctness proof consisted of a proof that $\mathscr{K}_+$ is faithful, which is a straightforward modification of the proof that that $\mathscr{K}$ is faithful.

4.5. "... and similarly for T_2". The idea of expanding the theory and then using $\mathscr{K}$ to interpret it worked so well for T_1 that trying to do the same for T_2 is the natural course of action. The first step, expanding the language of the more abstract desciption by adding the predicates Green_Channel, Red_Channel, Green_Connects, Red_Connects, and so on, is exactly the same. The only difference in this case is that the language $\mathscr{L}$ that is being expanded is not just the pure language of dataflow plus names of the objects that make up the particular architecture. It also contains some additional vocabulary introduced by the earlier implementation transformations. For example, it may contain the language of shared variables as a result of an earlier application of T_1 or T_2. The next step — conservatively extending the $\mathscr{L}$-theory Θ of the more abstract description to obtain $\mathscr{L}^+$-theory Θ^+ — is, again exactly the same for T_2 as for T_1. And, of course, sentences can be colored and Observation 1 continues to hold, just as in the case of T_1.

But, in this case, the theory Υ of the more concrete description analogously extended.[8] Just as the channel in the abstract description that the transformation will implement must be distinguished from those that are left alone, the new variable that is introduced in the concrete level description must be distinguished from those that were already present. Again, coloring will be employed, this time using Green_ for the freshly introduced variable — since it corresponds to the "green" channel — and Yellow_ for the pre-existing variables, and similarly for other types. Call the extended theory $\Upsilon^\times$.

Now define an interpretation $\mathscr{K}_\times$ of Θ^+ in $\Upsilon^\times$ by slightly modifying and extending the definition of $\mathscr{K}_+$.

$$
\begin{aligned}
\omega_{\mathscr{K}_\times} &= \mathsf{x}_0 \equiv \mathsf{x}_0\\
\mathscr{K}_\times(\mathsf{Procedure}) &= \mathsf{Procedure}(\mathsf{x}_0)\\
\mathscr{K}_\times(\mathsf{Channel}) &= \mathsf{Channel}(\mathsf{x}_0) \vee \mathsf{Variable}(\mathsf{x}_0)\\
\mathscr{K}_\times(\mathsf{Green\_Channel}) &= \mathsf{Green\_Variable}(\mathsf{x}_0)\\
\mathscr{K}_\times(\mathsf{Red\_Channel}) &= \mathsf{Channel}(\mathsf{x}_0)\\
\mathscr{K}_\times(\mathsf{Variable}) &= \mathsf{Yellow\_Variable}(\mathsf{x}_0)\\
\mathscr{K}_\times(\mathsf{In\_Port}) &= \mathsf{In\_Port}(\mathsf{x}_0,\mathsf{x}_1)\\
&\qquad \vee [\, \mathsf{Call}(\mathsf{x}_0,\mathsf{x}_1) \wedge \exists \mathsf{x}_2\, \mathsf{Can\_Get}(\mathsf{x}_0,\mathsf{x}_2)\,]\\
\mathscr{K}_\times(\mathsf{Green\_In\_Port}) &= \mathsf{Green\_Call}(\mathsf{x}_0,\mathsf{x}_1) \wedge \exists \mathsf{x}_2\, \mathsf{Green\_Can\_Get}(\mathsf{x}_0,\mathsf{x}_2)\\
\mathscr{K}_\times(\mathsf{Red\_In\_Port}) &= \mathsf{In\_Port}(\mathsf{x}_0,\mathsf{x}_1)\\
\mathscr{K}_\times(\mathsf{Call}) &= \mathsf{Yellow\_Call}(\mathsf{x}_0,\mathsf{x}_1)\\
\mathscr{K}_\times(\mathsf{Can\_Get}) &= \mathsf{Yellow\_Can\_Get}(\mathsf{x}_0,\mathsf{x}_1)\\
\mathscr{K}_\times(\mathsf{Out\_Port}) &= \mathsf{Out\_Port}(\mathsf{x}_0,\mathsf{x}_1)\\
&\qquad \vee [\, \mathsf{Call}(\mathsf{x}_0,\mathsf{x}_1) \wedge \exists \mathsf{x}_2\, \mathsf{Might\_Put}(\mathsf{x}_0,\mathsf{x}_2)\,]
\end{aligned}
$$

[8]The necessity of coloring Υ will be demonstrated below.

$$
\begin{aligned}
\mathscr{K}_{\times}\,(\mathsf{Green\_Out\_Port}) &= \mathsf{Green\_Call}(\mathsf{x}_0, \mathsf{x}_1) \wedge \exists \mathsf{x}_2\, \mathsf{Green\_Might\_Put}(\mathsf{x}_0, \mathsf{x}_2)\\
\mathscr{K}_{\times}\,(\mathsf{Red\_Out\_Port}) &= \mathsf{Out\_Port}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Might\_Put}) &= \mathsf{Yellow\_Might\_Put}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Can\_Carry}) &= \mathsf{Can\_Carry}(\mathsf{x}_0, \mathsf{x}_1) \vee \mathsf{Can\_Hold}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Green\_Can\_Carry}) &= \mathsf{Green\_Can\_Hold}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Red\_Can\_Carry}) &= \mathsf{Can\_Carry}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Can\_Hold}) &= \mathsf{Yellow\_Can\_Hold}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Might\_Supply}) &= \mathsf{Might\_Supply}(\mathsf{x}_0, \mathsf{x}_1) \vee \mathsf{Might\_Put}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Green\_Might\_Supply}) &= \mathsf{Green\_Might\_Put}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Red\_Might\_Supply}) &= \mathsf{Might\_Supply}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Might\_Put}) &= \mathsf{Yellow\_Might\_Put}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Can\_Accept}) &= \mathsf{Can\_Accept}(\mathsf{x}_0, \mathsf{x}_1) \vee \mathsf{Can\_Get}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Green\_Can\_Accept}) &= \mathsf{Green\_Can\_Get}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Red\_Can\_Accept}) &= \mathsf{Can\_Accept}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Can\_Get}) &= \mathsf{Yellow\_Can\_Get}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Connects}) &= \mathsf{Connects}(\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2)\\
&\qquad \vee [\, \mathsf{Writes}(\mathsf{x}_1, \mathsf{x}_0) \wedge \mathsf{Reads}(\mathsf{x}_2, \mathsf{x}_0)\,]\\
\mathscr{K}_{\times}\,(\mathsf{Green\_Connects}) &= \mathsf{Green\_Writes}(\mathsf{x}_1, \mathsf{x}_0) \wedge \mathsf{Green\_Reads}(\mathsf{x}_2, \mathsf{x}_0)\\
\mathscr{K}_{\times}\,(\mathsf{Red\_Connects}) &= \mathsf{Connects}(\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2)\\
\mathscr{K}_{\times}\,(\mathsf{Writes}) &= \mathsf{Yellow\_Writes}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathsf{Reads}) &= \mathsf{Yellow\_Reads}(\mathsf{x}_0, \mathsf{x}_1)\\
\mathscr{K}_{\times}\,(\mathbf{T}) &= \mathbf{T}(\mathsf{x}_0)\\
\mathscr{K}_{\times}\,(\mathbf{P}) &= \mathbf{P}(\mathsf{x}_0, \mathsf{x}_1, \ldots, \mathsf{x}_{n-1})\\
&\qquad\qquad \text{for every other predicate } \mathbf{P}\\
&\qquad\qquad \text{of } \mathscr{L}\text{, where } \mathbf{P} \text{ is } n\text{-ary}\\
\mathscr{K}_{\times}\,(\mathbf{c}) &= \mathsf{x}_0 \equiv \mathbf{v}\\
\mathscr{K}_{\times}\,(\mathbf{p}_{\mathrm{out}}) &= \mathsf{x}_0 \equiv \mathbf{k}_{\mathrm{write}}\\
\mathscr{K}_{\times}\,(\mathbf{p}_{\mathrm{in}}) &= \mathsf{x}_0 \equiv \mathbf{k}_{\mathrm{read}}\\
\mathscr{K}_{\times}\,(\mathbf{n}) &= \mathsf{x}_0 \equiv \mathbf{n} \qquad \text{for any name } \mathbf{n} \text{ of } \mathscr{L} \text{ other}\\
&\qquad\qquad \text{than } \mathbf{c},\ \mathbf{p}_{\mathrm{out}}, \text{ and } \mathbf{p}_{\mathrm{out}}
\end{aligned}
$$

Showing that $\mathscr{K}_{\times}$ faithfully interprets the colored theory of D'_1 in the colored theory of D'_2 is very much like showing that $\mathscr{K}_{+}$ faithfully interprets the colored theory of D'_0 in the theory of D'_1. For example,

$$
\begin{aligned}
\mathsf{Channel}^{\mathscr{K}_{\times}^{\partial}(\mathfrak{D}'_2)} &= \{x_0 \in |\mathfrak{D}'_2| : \mathfrak{D}'_2 \vDash \mathscr{K}_{+}\,(\mathsf{Channel})\,[x_0]\}\\
&= \{x_0 \in |\mathfrak{D}'_2| : \mathfrak{D}'_2 \vDash \mathsf{Channel}(\mathsf{x}_0) \vee \mathsf{Variable}(\mathsf{x}_0)\,[x_0]\}\\
&= \{v_1, v_2\}
\end{aligned}
$$

$$\begin{aligned}
\mathsf{Green_Channel}^{\mathscr{K}_\times^\partial(\mathfrak{D}'_2)} &= \{x_0 \in |\mathfrak{D}'_2| : \mathfrak{D}'_2 \models \mathscr{K}_+(\mathsf{Green_Channel})\,[x_0]\} \\
&= \{x_0 \in |\mathfrak{D}'_2| : \mathfrak{D}'_2 \models \mathsf{Green_Variable}(\mathsf{x}_0)\,[x_0]\} \\
&= \{v_2\}
\end{aligned}$$

$$\begin{aligned}
\mathsf{Red_Channel}^{\mathscr{K}_\times^\partial(\mathfrak{D}'_2)} &= \{x_0 \in |\mathfrak{D}'_2| : \mathfrak{D}'_2 \models \mathscr{K}_+(\mathsf{Red_Channel})\,[x_0]\} \\
&= \{x_0 \in |\mathfrak{D}'_2| : \mathfrak{D}'_2 \models \mathsf{Channel}(\mathsf{x}_0)\,[x_0]\} \\
&= \emptyset
\end{aligned}$$

$$\begin{aligned}
\mathsf{Variable}^{\mathscr{K}_\times^\partial(\mathfrak{D}'_2)} &= \{x_0 \in |\mathfrak{D}'_2| : \mathfrak{D}'_2 \models \mathscr{K}_+(\mathsf{Variable})\,[x_0]\} \\
&= \{x_0 \in |\mathfrak{D}'_2| : \mathfrak{D}'_2 \models \mathsf{Yellow_Variable}(\mathsf{x}_0)\,[x_0]\} \\
&= \{v_1\}
\end{aligned}$$

and so $\mathfrak{D}'_1$ and $\mathscr{K}_\times^\partial(\mathfrak{D}'_2)$ are indeed isomorphic. But note that if $\varUpsilon$ had not been colored, the proof would have broken down. The extension of Green_Channel in the image of $\mathfrak{D}'_2$ under the abstraction map $\mathscr{K}_\times^\partial$ would have been the set of all the shared variables of $\mathfrak{D}'_2$ (i.e., $\{v_1, v_2\}$) rather than just the set of "green" ones, $\{v_2\}$. Similarly, the extension of Variable in the image of $\mathfrak{D}'_2$ under the abstraction map would have been $\{v_1, v_2\}$ rather than just $\{v_1\}$.

All that remains to complete the proof of T_2's correctness is to define the interpretation $\mathscr{K}^\times$ of $\varTheta$ in $\varUpsilon$ and prove that, if $\mathscr{J}$ is a faithful interpretation of some pure dataflow theory $\varSigma$ in $\varTheta$, which was produced by composing some number of interpretations that correspond to transformations, then $\mathscr{K}^\times \circ \mathscr{J}$ is a faithful interpretation of $\varSigma$ in $\varUpsilon$. The pattern of our earlier example cannot slavishly followed, and $\mathscr{K}^\times(\varphi)$ defined as $\mathscr{K}_\times(\varphi^+)$, because this sentence is in the language of $\varUpsilon^\times$, not the language of $\varUpsilon$. Apparently, what is needed is some way of "uncoloring" $\mathscr{K}_\times(\varphi^+)$. The same sort of examples that showed coloring $\varUpsilon$ was necessary for faithfully interpreting $\varTheta^+$ show that $\varUpsilon^\times$ cannot naturally be faithfully interpreted in $\varUpsilon$, i.e., that there is no *general* method for faithfully uncoloring sentences in the language of $\varUpsilon^\times$. In fact, even a simple sentence of the form

$$\neg\mathsf{Yellow\_Variable}(\mathbf{a})$$

might be impossible to uncolor, since there are two ways **a** can fail to be a "yellow" channel, by being a non-channel and by being a "green" channel. But "yellow" and "green" channels are indistinguishable except with respect to color, so simply changing Yellow_Variable to either Variable or Channel without regard to **a** will violate faithfulness. Fortunately, faithfully uncoloring *every* sentence in the language of $\varUpsilon$ is not required. A restatement of the correctness criterion for $\mathscr{K}^\times$ will make this point clearer.

For any set $\varDelta$ of sentences in the language of $\varTheta$, and any mapping $\mathscr{I}$ of the sentences in the language of $\varTheta$ to sentences the language of in $\varUpsilon$, $\mathscr{I}$ will be said to be a *$\varDelta$-faithful interpretation of $\varTheta$ in $\varUpsilon$* if and only if, for every sentence φ in $\varDelta$,

$$\varphi \in \varTheta \iff \mathscr{I}(\varphi) \in \varUpsilon$$

The usual notion of faithfulness is special case in which $\varDelta$ is the set of all sentences in the language of $\varTheta$.

It follows directly from the definition of Δ-faithfulness that the following two conditions on faithful interpretation $\mathscr{J}$ of theory Σ in theory Θ and interpretation $\mathscr{I}$ of theory Θ in theory Υ are equivalent.

(1) $\mathscr{I} \circ \mathscr{J}$ is a faithful interpretation of theory Σ in theory Υ.

(2) $\mathscr{I}$ is a $\mathscr{J}[\Gamma]$-faithful interpretation of theory Θ in theory Υ, where Γ is the set of all sentences in the language of Σ.

So it is not necessary to faithfully uncolor *every* sentence in the language of $\Upsilon^\times$, only those sentences that are $\mathscr{K}_\times((\mathscr{J}(\psi))^+)$ for some dataflow language sentence ψ. It seems likely that coloring cannot affect whether a sentence $\mathscr{K}_\times((\mathscr{J}(\psi))^+)$ is a consequence of $\Upsilon^\times$: it has been shown that, for every sentence ψ in the language of Σ,

$$\begin{aligned} \psi \in \Sigma \quad &\Longleftrightarrow \quad \mathscr{J}(\psi) \in \Theta && (\mathscr{J} \text{ is faithful}) \\ &\Longleftrightarrow \quad (\mathscr{J}(\psi))^+ \in \Theta^+ && (\text{Observation 1}) \\ &\Longleftrightarrow \quad \mathscr{K}_\times\big((\mathscr{J}(\psi))^+\big) \in \Upsilon^\times && (\mathscr{K}_\times \text{ is faithful}) \end{aligned}$$

so whether a sentence mentioning "yellow" variables, "green" variables, and so on, is a consequence of $\Upsilon^\times$ or not is entirely determined by whether some other sentence that only talks about channels and connections is a member of Σ.

This intuition can be confirmed. For any sentence φ in the language of $\Upsilon^\times$, let φ^-, the *uncoloring* of φ, be the sentence in the language of Υ that results when all colored predicates are replaced by their uncolored counterparts. For example, both the predicate $\mathsf{Green\_Variable}$ and the predicate $\mathsf{Yellow\_Variable}$ are replaced by the predicate $\mathsf{Variable}$. By showing that, for every sentence ψ in the language of Σ,

$$\mathscr{K}_\times\big((\mathscr{J}(\psi))^+\big) \in \Upsilon^\times \iff \big(\mathscr{K}_\times\big((\mathscr{J}(\psi))^+\big)\big)^- \in \Upsilon$$

the proof that T_2 is correct, with its corresponding $\mathscr{J}[\Gamma]$-faithful interpretation $\mathscr{K}^\times$ defined by

$$\mathscr{K}^\times(\varphi) = (\mathscr{K}_\times(\varphi^+))^-$$

will be complete.

To make the structure of the proof will a bit clearer, an example, based on Figure 7 — which is similar to the familiar transformation of hierarchies in Figure 6, but with an additional channel that remains unimplemented — will be examined first. Consider a sentence of the form

$$\mathsf{Channel}(\mathbf{n})$$

where name $\mathbf{n}$ is a name that denotes a value in the domain of $\mathfrak{D}_0''$, the mathematical structure that corresponds to description D_0''. There are four cases to be considered.

Case 1: $\mathbf{n}$ *denotes the channel that is implemented by the application of* T_1.

In this case, coloring the sentence for the application of T_1 produces

$$\mathsf{Channel}(\mathbf{n}) \wedge \mathsf{Green\_Channel}(\mathbf{n})$$

which, according to the color constraints of the expanded theory of $\mathfrak{D}_0''$, is equivalent to

$$\mathsf{Green\_Channel}(\mathbf{n})$$

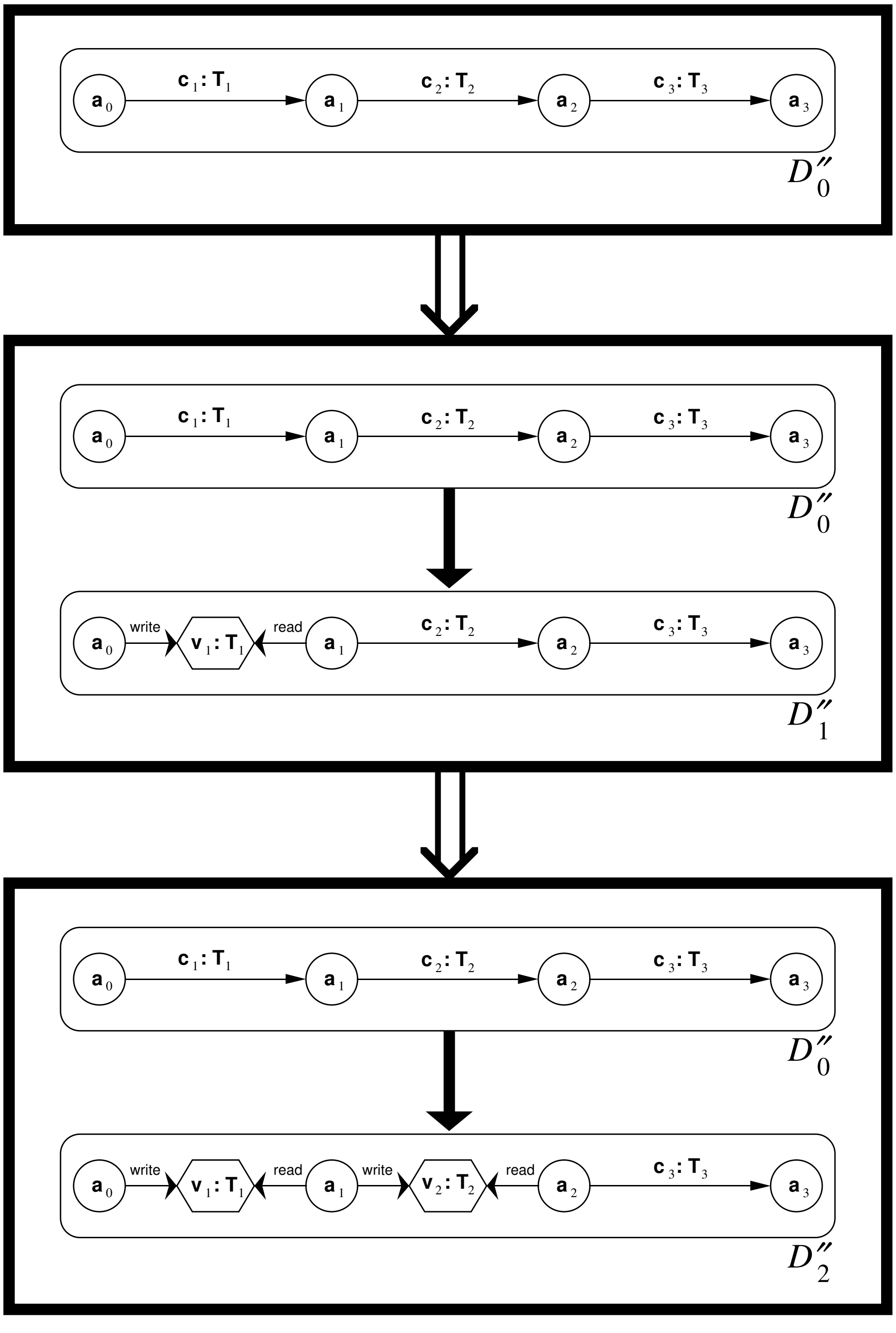

FIGURE 7. Incremental Replacement with Additional Channel

Applying $\mathscr{K}_+$ to the colored sentence results in

$$[\mathsf{Channel}(\mathbf{m}) \vee \mathsf{Variable}(\mathbf{m})] \wedge \mathsf{Variable}(\mathbf{m})$$

where name $\mathbf{m}$ denotes the implementing variable, which is equivalent, according to the type constraints of the theory of $\mathfrak{D}_1''$, to

$$\mathsf{Variable}(\mathbf{m})$$

Coloring this sentence for the application of T_2 has no effect, since it only adds a logically redundant conjunct. So applying $\mathscr{K}_\times$ yields a sentence equivalent to

$$\mathsf{Yellow\_Variable}(\mathbf{m})$$

which can be uncolored to yield

$$\mathsf{Variable}(\mathbf{m})$$

Case 2: $\mathbf{n}$ *denotes the channel that is implemented by the application of* T_2*.* Coloring the sentence for the application of T_1 produces

$$\mathsf{Channel}(\mathbf{n}) \wedge \mathsf{Red\_Channel}(\mathbf{n})$$

which, according to the color constraints of the expanded theory of $\mathfrak{D}_0''$, is equivalent to

$$\mathsf{Red\_Channel}(\mathbf{n})$$

Applying $\mathscr{K}_+$ to the colored sentence results in

$$[\mathsf{Channel}(\mathbf{n}) \vee \mathsf{Variable}(\mathbf{n})] \wedge \mathsf{Channel}(\mathbf{n})$$

which is equivalent, according to the type constraints of the theory of $\mathfrak{D}_1''$, to

$$\mathsf{Channel}(\mathbf{n})$$

Coloring this sentence for the application of T_2 results in

$$\mathsf{Channel}(\mathbf{n}) \wedge \mathsf{Green\_Channel}(\mathbf{n})$$

which is equivalent modulo the color constraints of the expanded theory of $\mathfrak{D}_1''$ to

$$\mathsf{Green\_Channel}(\mathbf{n})$$

Applying $\mathscr{K}_\times$ yields

$$[\mathsf{Channel}(\mathbf{m}) \vee \mathsf{Variable}(\mathbf{m})] \wedge \mathsf{Green\_Variable}(\mathbf{m})$$

where $\mathbf{m}$ denotes the implementing variable, which is equivalent modulo the color constraints of the expanded theory of $\mathfrak{D}_2''$ to

$$\mathsf{Green\_Variable}(\mathbf{m})$$

Uncoloring produces

$$[\mathsf{Channel}(\mathbf{m}) \vee \mathsf{Variable}(\mathbf{m})] \wedge \mathsf{Variable}(\mathbf{m})$$

which is equivalent modulo the type constraints of the theory of $\mathfrak{D}_2''$ to

$$\mathsf{Variable}(\mathbf{m})$$

Case 3: $\mathbf{n}$ *denotes the channel that remains unimplemented.*

Coloring the sentence for the application of T_1 again produces

$$\mathsf{Channel}(\mathbf{n}) \wedge \mathsf{Red\_Channel}(\mathbf{n})$$

or, equivalently,

$$\mathsf{Red\_Channel}(\mathbf{n})$$

and applying $\mathscr{K}_+$ to the colored sentence results in

$$[\mathsf{Channel}(\mathbf{n}) \vee \mathsf{Variable}(\mathbf{n})] \wedge \mathsf{Channel}(\mathbf{n})$$

or

$$\mathsf{Channel}(\mathbf{n})$$

just as in Case 2. Coloring for the application of T_2 also results in

$$\mathsf{Channel}(\mathbf{n}) \wedge \mathsf{Red\_Channel}(\mathbf{n})$$

or

$$\mathsf{Red\_Channel}(\mathbf{n})$$

Applying $\mathscr{K}_\times$ yields

$$[\mathsf{Channel}(\mathbf{n}) \vee \mathsf{Variable}(\mathbf{n})] \wedge \mathsf{Channel}(\mathbf{n})$$

or

$$\mathsf{Channel}(\mathbf{n})$$

Uncoloring is thus unnecessary.

Case 4: $\mathbf{n}$ *denotes a non-channel.*

In this case, the first coloring produces

$$\mathsf{Channel}(\mathbf{n}) \wedge \tau_{\mathbf{n}}$$

which is equivalent, modulo the color constraints of the expanded theory of $\mathfrak{D}_0''$, to a contradiction. Subsequent applications of $\mathscr{K}_+$, the second coloring, $\mathscr{K}_\times$, and the uncoloring all preserve this property of being equivalent, modulo general color and type constraints, to a contradiction.

In any case, uncoloring a sentence of the form

$$\mathsf{Channel}(\mathbf{n})$$

in $\Upsilon^\times$ yields a sentence in Υ, and uncoloring a sentence of this form not in $\Upsilon^\times$ yields a sentence not in Υ. This is because, at every stage, the sentence is either mapped to its corresponding implementing sentence or simply left alone if it is true, and is immediately eliminated as a candidate for inclusion in any implementing theory if it is false.

The proof that the same is true when the interpretation $\mathscr{K}^+$ is replaced by another interpretation $\mathscr{J}$ determined by transformations and the predicate $\mathsf{Channel}$ is replaced by another singulary predicate $\mathbf{P}$ is quite similar.

Case 1: $\mathbf{n}$ *has been implemented as* $\mathbf{m}$ *by an earlier transformation.*

Coloring $\mathscr{J}(\mathbf{P}(\mathbf{n}))$ for the application of T_2 adds conjuncts that are implied by Θ, such as $\tau_{\mathbf{m}}$, applying $\mathscr{K}_\times$ might add $\mathsf{Yellow\_}$ to some of the predicates of $\mathscr{J}(\mathbf{P}(\mathbf{n}))$

and some of the other conjuncts, but then uncoloring removes $\mathsf{Yellow\_}$ leaving $\mathscr{I}(\mathbf{P}(\mathbf{n}))$ conjoined with predicates that are implied by Υ. So,

$$\mathscr{I}(\mathbf{P}(\mathbf{n})) \in \Theta \iff (\mathscr{K}_\times((\mathscr{I}(\mathbf{P}(\mathbf{n})))^+))^- \in \Upsilon$$

in this case.

Case 2: $\mathbf{n}$ *denotes the channel that is implemented as variable* $\mathbf{m}$ *by the application of* T_2.

Coloring this sentence for the application of T_2 results in

$$\mathbf{P}(\mathbf{n}) \wedge \mathsf{Green\_Channel}(\mathbf{n})$$

which is equivalent modulo the color constraints of the expanded theory of $\mathfrak{D}_1''$ to

$$\mathsf{Green\_}\mathbf{P}(\mathbf{n})$$

Applying $\mathscr{K}_\times$ yields

$$\mathsf{Green\_}(\mathscr{K}(\mathbf{P}(\mathbf{n}))) \wedge \mathsf{Green\_Variable}(\mathbf{m})$$

where $\mathsf{Green\_}(\mathscr{K}(\mathbf{P}(\mathbf{n})))$ is the result of replacing every predicate $\mathbf{Q}$ of $\mathscr{K}(\mathbf{P}(\mathbf{n}))$ for which a predicate $\mathsf{Green\_}\mathbf{Q}$ has been introduced by $\mathsf{Green\_}\mathbf{Q}$. Uncoloring produces

$$\mathscr{K}(\mathbf{P}(\mathbf{n})) \wedge \mathsf{Variable}(\mathbf{m})$$

where the second conjunct is implied by Υ. So, by faithfulness of the original $\mathscr{K}$,

$$\mathscr{I}(\mathbf{P}(\mathbf{n})) \in \Theta \iff (\mathscr{K}_\times((\mathscr{I}(\mathbf{P}(\mathbf{n})))^+))^- \in \Upsilon$$

in this case as well.

Case 3: $\mathbf{n}$ *denotes a channel that remains unimplemented.*

Coloring for the application of T_2 also results in

$$\mathbf{P}(\mathbf{n}) \wedge \mathsf{Red\_Channel}(\mathbf{n})$$

or

$$\mathsf{Red\_}\mathbf{P}(\mathbf{n})$$

Applying $\mathscr{K}_\times$ yields

$$\mathscr{K}(\mathbf{P}(\mathbf{n})) \wedge \mathsf{Channel}(\mathbf{n})$$

Uncoloring is again unnecessary, and so once again

$$\mathscr{I}(\mathbf{P}(\mathbf{n})) \in \Theta \iff (\mathscr{K}_\times((\mathscr{I}(\mathbf{P}(\mathbf{n})))^+))^- \in \Upsilon$$

Case 4: $\mathbf{n}$ *denotes a non-channel.*

Either the type constraints of Σ require that the argument of $\mathbf{P}$ denotes a channel, or they forbid that the argument of $\mathbf{P}$ denotes a channel. If the former, there is equivalence to a contradiction after coloring, which is maintained by $\mathscr{K}_\times$ and uncoloring. If the latter, then $\mathscr{I}(\mathbf{P}(\mathbf{n})$ is unaffected by the application of T_2, and it will be a member of Υ iff it is a member of Θ. So, in this last case, as in all the others,

$$\mathscr{I}(\mathbf{P}(\mathbf{n})) \in \Theta \iff (\mathscr{K}_\times((\mathscr{I}(\mathbf{P}(\mathbf{n})))^+))^- \in \Upsilon$$

Essentially the same argument works for multinary predicates, and ultimately for sentences in general,[9] but the notation becomes more complicated and the number of cases to be considered increases. Ignoring details, it pretty much boils down to the fact that coloring has no effect on membership in the theories that correspond to the descriptions, because the conjuncts that are added due to coloring are all implied by the relevant theories, and so $\mathscr{K}^{\times}$ either implements according to $\mathscr{K}$ (which is faithful) or doesn't do anything (which is also faithful). Since this was our a priori reason for thinking T_2 is correct, the proof can be considered a formalization of that intuition.

Just as in the case of the proof of the correctness of T_1, this proof has several advantages over a proof produced by ad hoc modification of $\mathscr{K}$. First, there is no doubt that interpretation $\mathscr{K}^{\times}$ corresponds to T_2, since it was defined as applying $\mathscr{K}$ to the appropriately colored piece of the architecture. Second, the correctness of T_2 followed from the faithfulness of $\mathscr{K}$ in straightforward, albeit tedious, fashion.

5. Conclusions

5.1. The Main Results. Re-examining the definitions of $\mathscr{K}_+$ and $\mathscr{K}_\times$, the proof that $\mathscr{K}_+$ is faithful, and the proof that, if $\mathscr{J}$ is faithful, then so is $\mathscr{K}_\times \circ \mathscr{J}$, will reveal that the same constructions could be used with another interpretation $\mathscr{I}$ in place of $\mathscr{K}$, provided $\mathscr{I}$ corresponds to some simple implementation pattern[10] in the same way that $\mathscr{K}$ corresponds to the implementation pattern in Figure 1. Thus, what has actually been shown by the arguments above is

Result 1. If

(1) $\mathscr{I}$ is the faithful interpretation that corresponds to a correct simple implementation pattern in the same way that $\mathscr{K}$ corresponds to the refinement pattern in Figure 1,
(2) the architectural description in a maximal node in a correct architecture hierarchy contains a subarchitecture that matches the abstract description of the pattern, and
(3) the entire architectural description of the maximal node is in the same style as the abstract description of the pattern,

then the transformation rule that extends the hierarchy by adding a successor to the maximal node containing a description in which the subarchitecture is replaced by the concrete description of the refinement pattern (and all else remains unaltered) is correct, with corresponding interpretation $\mathscr{I}^+$.

and

Result 2. If

(1) $\mathscr{I}$ is the faithful interpretation that corresponds to a correct simple implementation pattern in the same way that $\mathscr{K}$ corresponds to the refinement pattern in Figure 1,

[9] More precisely, induction on formulas is employed. Variables are treated by assigning them values, formulas containing free variables are interpreted as if the variables were the names of those values. By generalizing over all assignments, the induction is completed.

[10] Intuitively, a simple implementation pattern is one that says a connector or component of some type can always be replaced by some structure composed of components and connectors.

(2) the architectural description in a maximal node in a refinement hierarchy contains a subarchitecture that matches the abstract description of the pattern,
(3) the predecessor of the maximal node contains a description with the same subarchitecture, and
(4) the entire architectural description of the predecessor is in the same style as the abstract description of the pattern,

then the transformation rule that modifies the hierarchy by replacing the description in the maximal node by a description in which the concrete description of the refinement pattern is substituded for the subarchitecture (and all else remains unaltered) is correct, with corresponding interpretation $\mathscr{I}^{\times}$.

This pair of results reduces the problem of verifying the pair of transformations that corresponds to a simple implementation pattern to verifying the pattern "in isolation", a problem that has already been shown to be quite feasible. The fundamental problems with the compositional approach — uncertainty whether the composite interpretation actually corresponds to the transformation and having to prove that a variety of modes of interpretation composition preserve faithfulness — have been completely avoided.

5.2. Increasing Formality. A question that the statement of the results and the arguments supprting them immediately raises is: Why are both the results and the arguments stated so informally? The reason is that they rely on an intuitive notion of what a refinement pattern is. The particular patterns used as illustrations were presented by drawing pictures of them, but these pictures have no precise semantics and so the arguments relied on an informal, intuitive understanding of what they meant.

In Sadl, the formal architectural description language we have developed [4], refinements can be thought of as having the form

An architectural description that matches pattern Π
and satisfies constraints Δ
can be refined to a description that matches pattern Π'
and satisfies constraints Δ'.

For example, one of our earlier papers [3, Pattern 3] contains the following refinement for merging shared variables: an archiectural description that matches the pattern[11]

```
@a0: Procedure[@@p00 -> @@p01]
@a1: Procedure[@@p10 -> @@p11]
@a2: Procedure[@@p20 -> @@p21]
@v1: Variable[@t]
@v2: Variable[@t]
@c0: ASSERTION = Writes(@a0, @v1)
@c1: ASSERTION = Reads(@a1, @v1)
@c2: ASSERTION = Writes(@a1, @v2)
```

[11]The notation is essentially self-explanatory once you know that terms that begin with "@" are pattern variables, the Sadl-equivalent of the metavariables in logical formulas, and that pattern variables that begin with "@@" are matched with collections of subtrees of the abstract syntax tree rather than individual subtrees.

```
@c3: ASSERTION = Reads(@a2, @v2)
@@rest
```

can be refined to a description that matches the pattern

```
@a0: Procedure[@@p00 -> @@p01]
@a1: Procedure[@@p10 -> @@p11]
@a2: Procedure[@@p20 -> @@p21]
@v:  Variable[@t]
@c0: ASSERTION = Writes(@a0, @v)
@c1: ASSERTION = Reads(@a1, @v)
@c2: ASSERTION = Writes(@a1, @v)
@c3: ASSERTION = Reads(@a2, @v)
@@rest
```

provided certain additional constraints on the architectural descriptions are satisfied.

One reason this pattern cannot be applied without requiring satisfaction of additional constraints is that collapsing the two variables into one might introduce an additional dataflow path from procedure $\mathbf{a}_0$ to procedure $\mathbf{a}_2$. However, this additional dataflow will not occur if, whenever $\mathbf{v}_1$ is written by $\mathbf{a}_0$, $\mathbf{v}_2$ is written by $\mathbf{a}_1$ before $\mathbf{v}_2$ is read by $\mathbf{a}_2$. A sufficient condition that guarantees this temporal property is that the procedures are run sequentially: first $\mathbf{a}_0$, next $\mathbf{a}_1$, and finally $\mathbf{a}_2$.[12] One way of adding this sequentially constraint to the refinement[13] is to include it as part of the concrete pattern, as follows

```
@a0: Procedure[@@p00 -> @@p01]
@a1: Procedure[@@p10 -> @@p11]
@a2: Procedure[@@p20 -> @@p21]
@v:  Variable[@t]
@c0: ASSERTION = Writes(@a0, @v)
@c1: ASSERTION = Reads(@a1, @v)
@c2: ASSERTION = Writes(@a1, @v)
@c3: ASSERTION = Reads(@a2, @v)
@c4: ASSERTION = Starts_After_Finish_Of(@a1, @a0)
@c5; ASSERTION = Starts_After_Finish_Of(@a2, @a1)
@@rest
```

With this modification, it is easy to prove that this refinement is correct *in isolation*, i.e., when `@@rest` is matched to the empty set in both patterns.

But it is easy to see that further restrictions are necessary when the pattern is applied in context. The problem is that procedures other than $\mathbf{a}_0$, $\mathbf{a}_1$, and $\mathbf{a}_2$ may be reading or writing $\mathbf{v}_1$ or $\mathbf{v}_2$ — e.g., one of the two variables may have been introduced by collapsing two other variables — and so collapsing $\mathbf{v}_1$ and $\mathbf{v}_2$ may introduce additional dataflow paths involving those other procedures. One or more

[12] Note that this assumes that Writes means *does write*, not *can write*.

[13] Explicitly adding constraints at the concrete level is quite appropriate if the refinement is begin used to generate a more concrete description from the abstract description, but is less appropriate if the objective is to check the correctness of existing hierarchies where the constraints may merely be implied by some other explicit constraints.

additional constraints, say

$$\forall \mathsf{x}\, [\, \mathsf{Procedure}(\mathsf{x}) \wedge \mathsf{Writes}(\mathsf{x}, \mathbf{v}_1) \rightarrow \mathsf{x} \equiv \mathbf{a}_0 \,]$$
$$\forall \mathsf{x}\, [\, \mathsf{Procedure}(\mathsf{x}) \wedge \mathsf{Reads}(\mathsf{x}, \mathbf{v}_1) \rightarrow \mathsf{x} \equiv \mathbf{a}_1 \,]$$
$$\forall \mathsf{x}\, [\, \mathsf{Procedure}(\mathsf{x}) \wedge \mathsf{Writes}(\mathsf{x}, \mathbf{v}_2) \rightarrow \mathsf{x} \equiv \mathbf{a}_1 \,]$$
$$\forall \mathsf{x}\, [\, \mathsf{Procedure}(\mathsf{x}) \wedge \mathsf{Reads}(\mathsf{x}, \mathbf{v}_2) \rightarrow \mathsf{x} \equiv \mathbf{a}_2 \,]$$

must be true of the abstract description to ensure that no additional dataflow will be introduced by the variable collapse.[14] It can now be shown that the refinement is correct, in the following sense: if an architectural description matches the first pattern above, implies the four additional conditions listed above, and is consistent with the order-of-execution constraints added to the second pattern, then it can be correctly refined to a description that matches the modified second pattern.

This example illustrates the sort of complexities that can arise in verifying refinement patterns more complicated than the simple implementation patterns considered in this paper. In particular, it shows that, in order to turn Results 1 and 2 into real theorems, a full formalization of refinements will be required, including a precise definition of *simple implementation pattern*, and that, as a result, the statement of the theorems is liable to be somewhat more complex than the informal statement of the results of this paper might suggest. If the ultimate goal is to automatically generate correct transformations from *any* refinement pattern that has been proven correct in isolation, some manner of automatically generating the additional conditions that must be satisfied is needed.

[14] These conditions are merely sufficient for correctness, not necessary.

References

1. M. Broy, *Compositional refinement of interactive systems*, to appear in the **Journal of the ACM**. (Available on the Web at `<http://www.jacm.org/papers/jacm1061.ps>`. An earlier version was published as Digital SRC Research Report 89.)
2. M. Moriconi and X. Quan, *Correctness and composition of software architectures*, **Proceedings of ACM SIGSOFT '94: Symposium on foundations of software engineering**, New Orleans, LA, December 1994, pp. 164–174. (Also available on the Web at `<http://www.csl.sri.com/sadl/fswe94.ps.gz>`.)
3. M. Moriconi, X. Quan, and R. A. Riemenschneider, *Correct architecture refinement*, **IEEE transactions on software engineering**, vol. 21 (1995), pp. 356–372. (Also available on the Web at `<http://www.csl.sri.com/sadl/tse95.ps.gz>`.)
4. M. Moriconi and R. A. Riemenschneider, **Introduction to SADL 1.0**, SRI Computer Science Laboratory Technical Report SRI-CSL-97-01, March 1997. (Available on the web at URL `<http://www.csl.sri.com/sadl/tse95.ps.gz>`.)
5. R. A. Riemenschneider, **A simplified method for establishing the correctness of architectural refinements**, SRI Technical Report SRI-CSL-97-??, ?? 1997. (Also available om the Web at `<http://www.csl.sri.com/sadl/simplified.ps.gz>`.)
6. W. M. Turski and T. S. E. Maibaum, **The specification of computer programs**, Addison-Wesley, 1987.

Computer Science Laboratory, SRI International, Menlo Park, CA 94025
E-mail address: rar@csl.sri.com