Property Transformations For Translations

Mirit Berg and Shmuel Katz
Computer Science Department
Haifa, Israel
{bmirit,katz}@cs.technion.ac.il
January 22, 2002

Abstract

When translating a system description from one specification language to another, it is, in many cases, inevitable that the underlying models of the original and the translated specification will be different. The notion of property transformation can be used to define the relation between the properties of the original and the translated model. In an earlier paper, definitions of 'faithful' translations were given, along with motivation and examples.

Here, a fuller theory of faithful property transformations is developed, including a calculus for combining translations and property transformations to obtain more complicated relations.

Property transformations are required to be independent of the source model, and this is shown to be a necessary restriction so that the transformation reflects the quality of the model translation. Sufficient conditions for proving the non-existence of property transformations are given. Generic translations are considered, derived directly from compilers that translate among formal methods model description languages. Property transformations for these translations are proven faithful for CTL*. The notion of an 'effective' source language is defined in terms of the logic that can be used to express properties in a tool associated with the target of the translation. This notion is then used to show that the most general transformation is not always the best.

Finally, compositions of model translation stages and, respectively, property transformations are considered. We prove that a composed property transformation is appropriate for sequentially composed model translations that each have faithful property transformations. Even when the translation of the model mixes translations of two previously analyzed stages non-sequentially, we define conditions when the simple composition of the property transformations is still faithful for the result, and prove the conditions for an example.

Keywords: translations among formal methods, property transformations, correctness, compositions of model translations.
1 Introduction

There are many different specification languages, either with or without accompanying verification tools. As noted in [6], there are several reasons why we might want to translate the specification of a certain system from one specification notation to another:

- The verification tools of different languages operate for different groups of properties. So to verify a certain property for a certain system, we might need to translate it to a specification language which has a verification tool which can operate on the property of interest.
- Even if a verification tool does permit verification for a certain kind of property, the verification of that property for a specific specification might fail, for example, because of an explosion in the size of the state space. On the other hand, another verification tool, which is appropriate for this case, might succeed.
- Some of the specification languages don’t have an accompanying verification tool at all. In this case, to automatically verify a property of the system, we must translate it to another specification language which is accompanied by such a tool.
- After verifying properties in the language of a formal methods tool, it may be necessary to translate the model to a notation which enables hardware synthesis and implementation.

Currently, in addition to translations between individual languages, there also exist more general translating systems, such as the SAL system [1], the Model Checking Kit [9] and Veritec [4]. These all allow translations for a variety of languages, and are extensible.

In all of the reasons shown above for translating a specification, the actual purpose is to learn about different properties of the system from its representation in different specification languages. In an ideal translation, the semantics of the system would be the same in the original representation and the translated one, and so its properties would be identical. The problem is that in real translations, because of the differences which exist between the different specification languages, it is likely that the semantics of the original and the translated model will not be identical. Examples of basic differences between modeling languages, and how they are reflected in changes to the semantic model during the translation, are shown in [6]. The differences in the semantics mean that the original and the result specifications (model descriptions) do not have the same properties.

Yet the process of checking a property for the original specification and deducing about the properties of the result specification, or the opposite way, can be done even if the semantic models of the two specifications are not identical. The idea is that although the models of the original and result specification are different, still there is a known connection between them which can be deduced from the translation. From this connection it is also possible to deduce a connection between the properties of the original and the result specifications.

This idea was formalized in the notion of a ‘faithful transformation’ defined
in [6]. Here, after briefly reviewing those definitions, a fuller theory of faithful property transformations is developed, extending the examples seen there.

Section 2 brings the formal definitions of the notions used. A transformation is defined to be a relation which defines the correspondence between the properties of the original and the result specification. An import-faithful transformation derives an automatically true property of the result model, from a known property of the source. A back-faithful transformation enables deducing a true property of the source from a property of the result, and a strongly-faithful transformation allows deducing properties in both directions.

In Section 3, the idea that the size of the class of properties for which there exists a faithful transformation for a certain translation can be used as an evaluation criterion for the quality of the translation, is made precise. In Section 3.1, we show that the property transformation must be independent of the source model in order to be a useful indicator of the quality of the model translation. In Section 3.2 a proof method is given that proves the non-existence of a strongly faithful property transformation for certain model translations and property classes.

Section 4 contains generic translations, and property transformations for them. The translations are generic in describing not a specific translation from one language to another, but a type of a change to the model. These are changes which are likely to be part of a large number of translations. The first translation (Section 4.1) adds intermediate states to the model, marking them as non-visible. Both the model translation and the property transformation are more general and more structured than previously seen, and a precise proof of strong faithfulness is given.

The second translation (Section 4.2) occurs when translating from a specification language which allows finite paths in the model it specifies to one which does not. To cope with this, the translation transforms the finite paths in the source model to infinite ones, by adding a self loop to the last states of the model (and marking them as 'Terminate'). A strongly faithful transformation for all CTL* properties is given for this translation.

The third example (Section 4.3) adds transitions which lead to 'dead end' paths. These are paths composed of states marked as 'fail'. A transformation for all CTL* properties is given, as well as transformations specific for the more restricted languages ACTL* and ECTL*. This is done in order to exemplify that the most general transformation is not always the best, and that the limitations of the verification tools involved should also be considered. To formalize this, we define the notion of an 'effective' source language. This is the sub-language of the source language for which the result properties are useful in a given context.

Finally, in Section 5, we show some results for translations which do several types of changes together to the model. The goal is to generate a transformation from the given property transformations for each of the separate changes. In Section 5.1 it is proved that a serial composition of two strongly faithful property transformations is strongly faithful for a model translation which is a serial composition of the corresponding translations. In Section 5.2 we show a translation which is an interleaved combination of two translations shown before. It
is proved that although the translation does not combine them in a serial composition, nevertheless, the transformation which is a serial composition of their two transformations, is strongly faithful for this translation.

2 Definitions

2.1 Programs, models, and translations

For every specification in a specification language, we can define a 'model' which describes its semantics.

The model, commonly known as a Kripke structure [3], is a directed graph with nodes that correspond to states over a state space of the values of the specification variables. The edges between the states reflect the possible changes in the variable values in the specification, due to some atomic action of the system. The actions possible in the specification language effectively limit the possible changes in states and thus the possible Kripke structures.

Temporal logic properties can be checked for the model, and from this we can draw conclusions about the properties of the specification.

Assume we have a translation between two specification languages, the first with a class of possible models $M_1$, and the second with a class $M_2$. We will say that $TR(M_1 \times M_2)$ is a translation-relation for this translation, if for every two models $M_1 \in M_1$ and $M_2 \in M_2$, $TR(M_1, M_2)$ if $M_2$ is a possible result of applying the translation to $M_1$.

$TR$ is always total over $M_1$, i.e., every possible source model has a translation.

2.2 Property transformations

As noted, a translation often changes the Kripke structure of the model, and thus changes also its temporal-logic properties. We want to define the connection between the properties of the original and the translated model. For this we use another relation which connects properties of the source and the translated model, defined over families of temporal logic properties.

We repeat here the definitions of faithful transformations, as seen in [6]:

Let $L_1$ and $L_2$ be two sub-languages of CTL*, and let $tr$ be a relation, $tr \subseteq L_1 \times L_2$, total for $L_1$. Also let $M_1$ and $M_2$ be two classes of models, and let $TR(M_1 \times M_2)$ be a translation relation. If for every two models $M_1 \in M_1$ and $M_2 \in M_2$ such that $TR(M_1, M_2)$ and for every two properties $\phi_1 \in L_1$ and $\phi_2 \in L_2$ such that $tr(\phi_1, \phi_2)$:

$M_1 \models \phi_1 \Rightarrow M_2 \models \phi_2$, then $tr$ is called an import-faithful property transformation for the translation $TR$.

$M_2 \models \phi_2 \Rightarrow M_1 \models \phi_1$, then $tr$ is called an back-(implication)-faithful property transformation for the translation $TR$.

$M_1 \models \phi_1 \iff M_2 \models \phi_2$, then $tr$ is called an strongly-faithful property transformation for the translation $TR$.
3 Quantifying the quality of a translation

When trying to assess the quality of a translation, the intuitive meaning is that a 'good' translation is one for which the model of the result program is as close as possible to that of the original program. (i.e., that the semantics of the original and result programs are similar). At first glance, it seems as if we can use property transformations to quantify this similarity.

If, for some translation, there exists a strongly faithful property transformation for a large group of properties, this seems to indicate that the original and result models are similar, because we can make a one-to-one translation between many of their properties. Thus a possible direction is to use the size of the group of properties for which there exists a strongly faithful property transformation as a measure for the quality of a translation. However, we will show that in order to do this, there needs to be a certain restriction on the allowed property transformations, or else even translations which are, intuitively, very 'low-quality', will be, by this measure, very 'high quality'.

3.1 A strongly faithful property transformation for a 'useless' translation

Even for a translation for which there is only a weak connection between the original and the result model, it is still possible to get a strongly faithful property transformation for all LTL properties by encoding enough information into the transformation.

Below we require only that the result model contains all the computations from the original model. Formally we say that a model translation relation $TR$ is linear complete if for two models $M_1$ and $M_2$, $TR(M_1, M_2) \Rightarrow M_2$ contains all the computations of $M_1$.

For example, the 'translation' which for every model $M_1$, generates the model which contains all the states as initial states, and from each state, all the other states as its successors (so it has a 'full computation tree') is a linear complete translation.

3.1.1 Representing the model as a formula

We assume the original model is given in a textual notation by $-I(s) :$ a formula for a state, true iff it is an initial state of the system.

$-T_1(s, s'), T_2(s, s'), \ldots, T_n(s, s') :$ a series of transition functions, which represent which transitions the system is allowed to make from one state to the next. Also we assume that for each $T_i$, the function $Enable(T_i)(s)$ indicates whether the $T_i$ transition is enabled in state $s$.

Each transition is assumed to be deterministic, with at most a single next state for each state (depending on whether or not that transition is enabled on that state). Non-determinism is handled by having many transition functions.

To build the property transformation for LTL properties we start by getting from the system definitions a formula which will hold for exactly the set of
computation paths of the system.

One possibility is to write a temporal logic formula in the following way:
\[ \Phi = I \land G \left[ \exists T_i \left( \text{Enabled}(T_i) \land \exists \bar{x} \left( \bar{x} = \bar{x} \land X T_i(\bar{x}, \bar{x}) \right) \right) \right] \]
where \( \bar{x} \) always stands for the current state, so it is a 'flexible' variable, as defined in [8]. \( \bar{x} \) is just used to save the current state in order to later check it with respect to the following one.

The formula demands that in each state in the path, the current state and the next state are connected by one of the \( T_i \) relations enabled in the state.

Note - Lamport has developed a logic called 'TLA', presented in [7], designed specifically to specify the behavior of a given system (actually its computation paths) and also its properties. We could use TLA to specify the set of possible executions of the original model, instead of the way which we saw here.

Now we can define \( tr \) for every LTL property \( \Psi \) to be: \( tr(\Psi) \equiv \Phi \rightarrow \Psi \)

Note that even for an arbitrary model translation (where nothing is known about the connection between the original and the result model) \( tr \) is always import faithful. This is true because if we know that in the original model, all the computations satisfy \( \Psi \), then all those computations in the result model which are also computations from the original model (indicated by their satisfying \( \Phi \)) satisfy \( \Psi \), and this is all that \( tr \) claims.

But for an arbitrary model translation, \( tr \) is not necessarily back-faithful. In particular, the result model might not contain some of the computations from the original model, and for these computations \( \Psi \) might not hold, although \( tr(\Psi) \) holds for the result model. In the extreme case, the result model might not contain any calculations from the original, and then it will always satisfy \( tr(\Psi) \) vacuously, without any connection to whether or not the original satisfies \( \Psi \).

However, for every linear complete model translation \( TR \), the property transformation \( tr \) is strongly faithful. Thus we managed to get a strongly faithful transformation for all LTL properties, although the limitation on the model translation is much too weak to be called a 'translation' in the intuitive sense. Even the 'translation' which simply develops the whole computation tree, which is surely not a translation in the intuitive sense, meets these requirements.

From the discussion above it is clear that in order to use the existence of a strongly faithful property transformation as a measure of the quality of a translation, we have to limit the allowed property transformations so that they do not use information from the original model.

### 3.2 Proving the non-existence of a strongly faithful transformation

In some cases, it may be impossible to find strongly faithful property transformations. One case where this occurs can be called 'information-losing translations'.

**Claim** - If for a translation \( TR \), there exist two source models \( M_1 \) and \( M_2 \) which have the same result model \( M \), and for some property \( \phi \) it is true that \( M_1 \models \phi \), but \( M_2 \not\models \phi \) then there cannot exist a strongly faithful property transformation for \( \phi \), for the translation \( TR \).
proof - Assume there does exist a strongly faithful property transformation \( tr \). By the definitions, \( M_1 \models \phi \Rightarrow M \models tr(\phi) \Rightarrow M_2 \models \phi \) and this is a contradiction to the assumption.

Note that, on the other hand, information losing translations can be either import-faithful or back-implication faithful. In the former case, both \( tr(\phi) \) and \( tr(\neg \phi) \) must be \( \text{true} \), while for back-implication both \( tr(\phi) \) and \( tr(\neg \phi) \) must be \( \text{false} \) (and thus not restricted for the original models), but in either case, this is not a contradiction.

Example: Consider the class of translations which replace edges in the original model with a sequence of 'intermediate' states in the result model, but without marking these intermediate states with any particular label which allows to differentiate them from the states of the original model.

Claim - For any translation which adds intermediate states which cannot be differentiated from the states of the original model, there cannot exist a strongly faithful transformation for the property \( Gp \), for any atomic property \( p \).

The claim can be demonstrated for the three models \( M_1 \), \( M_2 \) and \( M_3 \) shown in Figure 1. In these models - \( M_3 \) can be the result of translating both \( M_1 \) and \( M_2 \) (by adding intermediate states to them), while \( M_1 \models Gp \) and \( M_2 \nmodels Gp \). Thus we have two models (\( M_1 \) and \( M_2 \)) which have the same translated model (\( M_3 \)). Yet one satisfies the formula \( Gp \) and the other doesn’t. So we can use the general proof for the non-existence of a strongly faithful transformation, to conclude that for translations of this sort, there is no strongly faithful property transformation for the property \( Gp \).

4 Generic translations and property transformations for them

4.1 Adding intermediate states, marked as non-visible

Consider a translation where a single action in the source is divided into several target actions, due to different grains of atomicity. Translations in this family are called 'refinement translations'. Thus the target model will contain intermediate states between the states of the original model. Also we assume that the result program has an additional flag(state component) called 'Visible' which is turned on when the system is in a state from the original model\(^1\), and turned off when it is in one of the intermediate states.

Definition - A path where all the states except the first and the last have a false value for their Visible flag, will be called an intermediate path.

In a generic translation which does such refinement, the result model is characterized by having: all of the state variables from the original model, plus an additional 'Visible' flag; all the states of the original model, with a true value for the 'Visible' flag;

\(^1\)In all the translations, we refer to a state from the original model, and the state corresponding to it in the result model as the same state, although they are not exactly the same - in this example, the state from the result model has the additional Visible flag, which is true.
additional states, which have a false value for their 'Visible' flag, and which satisfy the following conditions:
1. For every two states which were connected by an edge in the original model, there exists at least one intermediate path between them in the result model.
2. For every two states which were not connected by an edge in the original model, there is no intermediate path between them in the result model.
3. There are no loops of only non-visible states (and thus there cannot be an infinite sequence of only non-visible states in the model paths).
4. In the paths of the result model the non-visible states always must appear as a finite sequence between visible states and not at the end of a path (This
demand is a consequence of the previous one when the result model contains only infinite paths).
Note that we do not demand here that the non-visible intermediate paths for different pairs of states are distinct. Different intermediate paths can share the same non-visible states. Also, there can be several intermediate paths instead of one original edge.

4.1.1 The property transformation

We define a property transformation for CTL*, and then prove that it is strongly faithful for all refinement translations. The transformation, \( tr \), will be defined by an induction on the structure of the formula.

* for \( \phi = p \) an atomic proposition: \( tr(p) = p \)
* \( tr(\neg \phi_1) = \neg tr(\phi_1) \)
* \( tr(\phi_1 \lor \phi_2) = tr(\phi_1) \lor tr(\phi_2) \)
* \( tr(\phi_1 \land \phi_2) = tr(\phi_1) \land tr(\phi_2) \)
* \( tr(X \phi_1) = X[\text{Visible} \ U [\text{Visible} \lor tr(\phi_1)]] \)
* \( tr(G \phi_1) = G[\text{Visible} \rightarrow tr(\phi_1)] \)
* \( tr(\phi_1 U \phi_2) = [\text{Visible} \rightarrow tr(\phi_1)] \ U [\text{Visible} \lor tr(\phi_2)] \)
* \( tr(A \phi_1) = A tr(\phi_1) \)
* \( tr(E \phi_1) = E tr(\phi_1) \)

4.1.2 Proof that \( tr \) is strongly faithful

Indicate by \( TR \) any refinement translation. We want to prove that \( tr \) is strongly faithful for this translation, so that for every CTL* formula \( \phi \), and for every two models \( M_1 \) and \( M_2 \), if \( TR(M_1, M_2) \) is true, then \( M_1 \models \phi \iff M_2 \models tr(\phi) \). We first introduce a definition and prove a lemma.

Definition - We say that a path \( \pi_2 \) from \( M_2 \) is a ‘stretch’ of a path \( \pi_1 \) from \( M_1 \), if \( \pi_2 \) is the same as \( \pi_1 \) except from that all the states from \( \pi_1 \) are marked as visible, and every edge is replaced with a finite sequence of states, marked as non-visible.

Lemma - For every CTL* path formula \( \phi \), and for every two paths \( \pi_1 \) and \( \pi_2 \), such that \( \pi_2 \) is a stretch of \( \pi_1 \), it is true that \( \pi_1 \models \phi \iff \pi_2 \models tr(\phi) \).

Note that when referring to a path formula here, the intention is any formula composed of a mixture of all CTL* operators and quantifiers. The formula is checked for a path, but in the context of the model. In the specific case where the formula is also a state formula, then a path satisfies it iff the first state in the path satisfies it.

proof of the lemma - (by an induction on the structure of \( \phi \))

base: for \( \phi = p \) an atomic proposition - because \( \pi_2 \) is a stretch of \( \pi_1 \) then in particular it starts with the same state, so \( \pi_1 \models p \iff \pi_2 \models p \), and since \( tr(p) = p \), \( \pi_2 \models tr(\phi_1) \) (because of the induction assumption) \( \iff \pi_2 \models tr(\phi_1) \iff \pi_2 \models tr(\neg \phi_1) \)
For the next cases we first notice the following fact: $\pi_2$ is a stretch of $\pi_1$, and so by the definition of a stretched path, it is also true that for every initial state of the original and the result model are the same. So we can conclude that:

$M_1 \models \phi \iff \text{for every initial state } s \text{ of } M_1, M_1 s \models \phi \iff \text{for every initial state } s \text{ of } M_2, M_2 s \models tr(\phi) \iff M_2 \models tr(\phi)$. q.e.d.
4.2 Property transformation for the translation which transforms finite paths to infinite

4.2.1 Motivation

Some languages allow describing a model with finite paths. For example, in a textual transition system, if the system reaches a state in which none of the transitions is enabled, then this will be the termination state of that path, and the path will be a finite one. In other languages, like SMV, there is a default transition which is always enabled, and so all the computation paths of the model are infinite (this is something that characterizes languages which are used to describe hardware behavior, because hardware, in contrast to a program, doesn’t ‘stop’).

4.2.2 The translation

The way in which a concrete translation from a transition system notation to SMV handles this problem is the following: the result model has all the state variables from the original model, plus an additional one - a boolean flag called ‘Terminate’. The result model contains all the states and edges from the original model, where for all the states, except for those which are the termination states of paths in the original model, the value of the terminate flag is false, and for those which are termination states of paths in the original model, the value of the terminate flag is true. There are also additional edges - all the termination states in the original model have additional self loops in the target, in addition to having their terminate flag true. The result is that every finite path in the original model is replaced in the result model with a path in which the last state is duplicated infinite times, with all the copies of the last state marked as terminate states.

4.2.3 definitions of temporal logic properties on finite paths

The temporal logic operators are usually defined for infinite paths. So first of all we will redefine them for finite paths.
* The definition of the F,G,U operators stays almost the same, only instead of being defined on an infinite range of indexes, they will be defined for a finite one. For instance, for the 'F' operator the definition will be:
  \[ M, \pi \models F \phi_1 \iff \text{there exists a } 0 < k \leq n \text{ such that } \pi^k \models \phi_1, \text{ where } n \text{ is the length of } \pi. \] (instead of just \( k > 0 \) as in the infinite definition)
  and in a similar way for the 'G' and 'U' operators.

* The definition for the boolean operators, and for the 'A' and 'E' quantifiers stays exactly as it was for the finite case.

* The definition of the 'X' operator is the only one which changes significantly. The new definition will be:
  - if \( \pi \) has more than a single state then \( M, \pi \models X \phi_1 \iff M, \pi^1 \models \phi_1 \)
  - if \( \pi \) has only a single state then \( M, \pi \not\models X \phi_1 \) (without any conditions)
A result of the last definition is that termination states can be identified by being the only ones that satisfy the formula \( \neg \text{Xtrue} \).

### 4.2.4 The property transformation

For every CTL* formula \( \phi \), we will define \( \text{tr}(\phi) \) by an induction on the structure of \( \phi \):

- For a formula starting with an 'X' operator:
  \[
  \text{tr}(\text{X} \phi_1) = \text{X}[\text{tr}(\phi_1)] \land \neg \text{Terminate}
  \]

  \( \text{tr} \) affects only the 'X' operator. It does not affect atomic formulas, and the rest of the operators and quantifiers, only 'pushes' it to the inner formula. Applying \( \text{tr} \) on the rest of the operators and quantifiers, only 'pushes' it to the inner formula. Meaning that for the unary operators \( \text{op=}, \text{G, A, E} \):
  \[
  \text{tr}(\text{op}(\phi_1)) = \text{op}(\text{tr}(\phi_1))
  \]

  And for the binary operators \( \text{op=} \land, \lor, \land, \lor, \land, \lor \):
  \[
  \text{tr}(\phi_1 \text{op} \phi_2) = \text{tr}(\phi_1) \text{op} \text{tr}(\phi_2)
  \]

  The proof that this transformation is indeed strongly faithful, is similar to the one shown for the transformation which adds intermediate non-visible states.

### 4.3 The model translation which adds 'Fail' states

The result model is the same as the original model, with an addition of fail states and transitions to them. Fail states are indicated by a special atomic proposition - 'Fail' which is \textit{true} for them, and \textit{false} for all other states. Once the system reaches a fail state, all the states from then on will be fail states (all the sub-tree under it). In a view of the result model as a Kripke structure, we can look at it as having a single fail state, which is a sink (has only a self loop).

More formally, the result model contains all the states and the transitions between then from the original model, where for all these states the value of the 'Fail' flag is \textit{false}. In addition the system may have new states, all of which are marked with a \textit{true} 'Fail' flag. The original, non-fail\footnote{From now on we will call states marked with \textit{true} 'Fail' flag fail states, and states with \textit{false} 'Fail' flag, non-fail states} states, may have, in addition to their sons from the original model, new sons which are fail states. For all the fail states, all their sons must be also fail states.

The initial states of the result model are the same as those in the original model, so in particular, they are all non-fail states.

The motivation for such a model translation is that the original program contains transitions defined by a relation (between the state before and after the transition) that cannot be expressed in the result language, so the translation builds the corresponding transition in the result model so that it has extra 'next states' that may not satisfy the needed relation (in the worst case this could be a totally non-deterministic assignment from the state space). In a specific execution, one value from the extended assignment is chosen, and then the system checks the relation condition with respect to the previous state. If the relation is not satisfied, the global 'Fail' flag of the system is turned on, and stays on throughout the rest of this calculation. Therefore this state, as well the
following ones, are marked as fail states. Because of this behavior, the result model looks as described above (with all the states not in the original model being 'Fail' states).

4.3.1 A strongly faithful transformation for all CTL* properties

For every CTL* formula, \( \phi \) we will define \( tr(\phi) \) by an induction on the structure of \( \phi \):

\[
\begin{align*}
tr(A\phi_1) &= A[(G\neg\text{Fail}) \rightarrow tr(\phi_1)] \\
tr(E\phi_1) &= E[(G\neg\text{Fail}) \land tr(\phi_1)]
\end{align*}
\]

\( tr \) affects only the 'A' and 'E' quantifiers. It does not affect atomic formulas, and the rest of the operators. Applying \( tr \) on the rest of the operators, only 'pushes' it to the inner formula, as described for the transformation for the translation which transforms finite paths to infinite ones.

The proof that this transformation is indeed strongly faithful is similar to the one shown for the transformation for the translation which adds intermediate non-visible states.

We will use this example to show another point concerning the use of transformations.

4.3.2 Limitations on the target language

Many times, the reason that we want a strongly faithful transformation is to verify the transformed properties on the translated model, with a verification tool associated with the target specification language, and from that deduce something about the original model. That is, ultimately we often want to use back-implication, even if import faithfulness is also used.

However, often the tool of the target specification language can only operate for some sub-language of CTL*. If this is the case then we will not be able to use back-implication for all the properties in the source language of the transformation, but only those with a transformation result in the language of properties on which the tool of the target specification language can operate.

Assume we are using a property transformation \( tr \) defined for a source language \( L_1 \) (for simplicity, we assume here that \( tr \) is a function), together with a translation to some model specification language, which has a verification tool that can verify properties from some language \( L_* \). We will define the 'effective source language' to be all the properties \( \phi \) from \( L_1 \) such that \( tr(\phi) \in L_* \).

When using a transformation with a translation to a specific language, then often, what we really want to maximize, is not the source language of the transformation, but the effective source language.

It may be the case that we have two different strongly faithful transformations for the same translation, with different source languages (groups of properties). Now we see that the one with the larger source language is not necessarily the better one, because it may have a smaller effective source language. We will demonstrate this for our current translation (which adds 'Fail' states).
For this translation we have just seen a strongly faithful property transformation whose source language is all \(\text{CTL}^*\). Now we will see two different strongly faithful property transformations for this translation, which have smaller source languages.

\(\text{ACTL}^*\) ([2],[5]) is the sub-language of \(\text{CTL}^*\), with only 'A' quantifiers (no 'E's). To prevent the creation of 'Exists' with negation, negation is allowed only on atomic propositions. To compensate for the loss of expressive power because of the lack of negation, we use both conjunction and disjunction (with negation we would need only one of them) and also both Until and Release temporal operators.

The meaning of the Release operator is:

\[
\pi \models \phi_1 \mathcal{R} \phi_2 \iff \text{for all } j \geq 0, \text{ if for every } i < j \text{ s.t. } \pi^i \not\models \phi_1 \text{ then } \pi^j \models \phi_2
\]

A transformation for \(\text{ACTL}^*\) properties
For every \(\text{ACTL}^*\) formula, \(tr_1\) does the following:
- Every atomic formula \(p\), without negation, is replaced with \(p \land \text{Fail}\)
- Every negation of an atomic formula, \(\neg p\), is replaced with \(\neg p \lor \text{Fail}\)
The rest of the formula stays as it was.

\(\text{ECTL}^*\) ([2],[5]) is the sub-language of \(\text{CTL}^*\), with only 'E' quantifiers. Here too negation is allowed only on atomic propositions, and the compensation for the loss of expressive power because of this is done in the same way.

A transformation for \(\text{ECTL}^*\) properties
For every \(\text{ECTL}^*\) formula, \(tr_2\) does the following:
- Every atomic formula \(p\), without negation, is replaced with \(p \lor \text{Fail}\)
- Every negation of an atomic formula, \(\neg p\), is replaced with \(\neg p \land \text{Fail}\)
The rest of the formula stays as it was.

Note that these transformations are 'local' in that only the atomic formulas are affected, and the rest of the formula is unaffected.

Again we will skip the proof for these two transformations, because it too is similar to the proof for the transformation of the translation which adds intermediate states.

Assume that we are using this translation (addition of 'Fail' states) to translate to a specification language which has a related tool which operates on properties from \(\text{CTL}^*\) only (as is true, for example, for the SMV specification language). Let's check what will be the effective source languages for the transformations \(tr, tr_1,\) and \(tr_2\) which we have seen before.

- for the transformation \(tr\) (the transformation for all \(\text{CTL}^*\)) - If we apply this transformation on a property which contains an 'A' quantifier, then the result property will contain a unit of the form - \(A [(G \neg \text{Fail}) \rightarrow \phi] \) (for some property \(\phi\)). Every property which contains such a unit is not a \(\text{CTL}^*\) property (because the operators here are not of the form of a pair of a quantifier and a temporal operator, as is demanded in \(\text{CTL}\)). In the same way we can see that for every property which contains an 'E' quantifier, the result property will not be a \(\text{CTL}\) property. The conclusion is that all the properties which contain a quantifier (either an 'A' or an 'E') are not in the effective source language.

Also, any \(\text{CTL}^*\) property which does not contain quantifiers, also cannot contain any of the path temporal operators (because all \(\text{CTL}^*\) properties are
state properties). So the properties in the effective source language cannot contain temporal operators either, i.e., the properties in the effective source language can only contain logical operators. In addition, we can see from the definition of \( tr \), that for any property which contains only logical operators, the result property is the same as the property itself. And because every property which contains only logical operators is in CTL, then the final result is that the effective source language for \( tr \) in this case is all the properties which contain only logical operators.

- for the transformation \( tr_1 \) (the transformation for ACTL\(^*\)) - This transformation maintains the structure of the source property in the result property (the only change is on the level of the atomic properties). So the result property is in CTL iff the source property is in CTL, i.e., for this transformation the effective source language is the intersection of CTL with the transformation source language which is ACTL\(^*\), so it is all ACTL.

- for the transformation \( tr_2 \) (the transformation for ECTL\(^*\)) - by a similar analysis the effective source language for this transformation in this case is ECTL.

We see that if we are indeed translating to a language which has a tool which operates only on CTL properties, then the transformations from ACTL\(^*\) and ECTL\(^*\) have a larger effective source language than the transformation from all CTL\(^*\). In this case we got that although \( tr_1 \) and \( tr_2 \) have smaller source languages, still, they have larger effective source languages. In general it is not always right to insist on getting a transformation with a source language which is as large as possible. Often, if we compromise on the size of the source language, we can get a transformation with a larger effective source language.

However, all this does not mean that the transformation from all CTL\(^*\) is generally useless. For example, if the target language is CTL\(^*\), then the effective source language for the transformation for all CTL\(^*\), is CTL\(^*\). Also the transformation for CTL\(^*\) transforms LTL properties to other LTL properties, so the effective source language would be all LTL if the target language is LTL.

5 Composition of Translations and Transformations

So far, we presented several examples of generic translations, where each does a specific change to the model, and for each we saw a property transformation. But what about translations which do several changes together? Do we have to analyze the translation as a new case, to try to find the property transformation, or can we use the property transformations which we already have for the separate translations, and combine them?

5.1 Serial Composition

We first look at the simple case where the new translation operates by doing two known translations serially, one after the other.
For translations and property transformations defined by a relation we will define a composition in the same way in which composition of relations is defined. For two translations $TR_1$ and $TR_2$, the composed translation $TR_2 \cdot TR_1$ will be defined as:
for every two models $M_1$ and $M_3$, $TR_2 \cdot TR_1 (M_1, M_3) \leftrightarrow$ there exists a model $M_2$ such that $TR_1 (M_1, M_2)$ and $TR_2 (M_2, M_3)$
In a similar way, for every two properties $\phi_1$ and $\phi_3$, $tr_2 \cdot tr_1 (\phi_1, \phi_3) \leftrightarrow$ there exists a property $\phi_2$ such that $tr_1 (\phi_1, \phi_2)$ and $tr_2 (\phi_2, \phi_3)$
Suppose we have two translations: one defined by a relation $TR_1 \subseteq (M_1 \times M_2)$ with a strongly faithful property transformation relation $tr_1 \subseteq (L_1 \times L_2)$, and another defined by a relation $TR_2 \subseteq (M_2 \times M_3)$ with a strongly faithful property transformation relation $tr_2 \subseteq (L_2 \times L_3)$.
So we have the two following facts:
1. For every two models $M_1 \in M_1$ and $M_2 \in M_2$ such that $TR_1 (M_1, M_2)$, and for every two properties $\phi_1$ in $L_1$ and $\phi_2$ in $L_2$ such that $tr_1 (\phi_1, \phi_2)$:
   $M_1 \models \phi_1 \Rightarrow M_2 \models \phi_2$
2. For every two models $M_2 \in M_2$ and $M_3 \in M_3$ such that $TR_2 (M_2, M_3)$, and for every two properties $\phi_2$ in $L_2$ and $\phi_3$ in $L_3$ such that $tr_2 (\phi_2, \phi_3)$:
   $M_2 \models \phi_2 \Rightarrow M_3 \models \phi_3$

5.1.1 Making the composition

Claim - Given $(TR_1, tr_1)$ a strongly faithful pair for the model classes $M_1$ and $M_2$ and property languages $L_1$ and $L_2$, and $(TR_2, tr_2)$ a strongly faithful pair for the model classes $M_2$ and $M_3$, and property languages $L_2$ and $L_3$, as defined above, then for every two models, $M_1$ and $M_3$, and two properties $\phi_1$ and $\phi_3$, if the following conditions are met:
1. $M_1 \in M_1$ and $M_3 \in M_3$.
2. there exists a model $M_2 \in M_2 \cap M_2$ such that $TR_1 (M_1, M_2)$ and $TR_2 (M_2, M_3)$.
3. $\phi_1 \in L_1$ and $\phi_3 \in L_3$.
4. there exists a property $\phi_2 \subseteq L_2 \cap L_3$ such that $tr_1 (\phi_1, \phi_2)$ and $tr_2 (\phi_2, \phi_3)$.
Then it is true that $M_1 \models \phi_1 \Rightarrow M_3 \models \phi_3$

Proof - Given all the assumptions of the claim for three models $M_1$, $M_2$, $M_3$ and three properties $\phi_1$, $\phi_2$ and $\phi_3$ (where, as above, $M_2$ and $\phi_2$ are the intermediate model and property as promised by the assumptions of the claim), we want to prove that $M_1 \models \phi_1 \Rightarrow M_3 \models \phi_3$.
It is true that $M_1 \models \phi_1 \Rightarrow M_2 \models \phi_2$ (because we assumed that $tr_1$ is strongly faithful for $TR_1$, and by our assumptions $M_1$, $M_2$, $\phi_1$, and $\phi_2$ satisfy all the needed conditions for this transformation).
In a similar way we also know that $M_2 \models \phi_2 \Rightarrow M_3 \models \phi_3$.
By combining these two facts we have $M_1 \models \phi_1 \Rightarrow M_3 \models \phi_3$.

Corollary - the transformation $tr_2 \cdot tr_1 \subseteq (L_1 \times L_3)$ is strongly faithful for the translation $TR_2 \cdot TR_1 \subseteq (M_1 \times M_3)$, so that if $TR_2 \cdot TR_1 (M_1, M_3)$ and $tr_2 \cdot tr_1 (\phi_1, \phi_3)$ then $M_1 \models \phi_1 \Rightarrow M_3 \models \phi_3$. 

16
proof - If we know that the models $M_1$ and $M_3$ are in the relation $TR_2-TR_1$, then demands 1 and 2 of the claim are met for them. Also if we know that the properties $\phi_1$ and $\phi_3$ are in the relation $tr_2-tr_1$, then demands 3 and 4 of the claim are met for them ($\phi_1$, $\phi_3$, and the intermediate property $\phi_2$ have to be in the languages specified in the claim because we know that the property transformation relations $tr_1$ and $tr_2$ are well defined for them, and the same for the models). So together we get that all four demands of the claim are met, and so by the claim, $M_1 \models \phi_1 \Leftrightarrow M_3 \models \phi_3$. q.e.d.

* The same analysis would be true if both transformations are import faithful, or both are back faithful. Of course, in that case we would get that the composed transformation is also import, or back faithful, respectively.

* Note that the composition also holds when the translation, or the property transformation, or both, are defined as functions instead of relations, since that is simply a special case.

Next we will demonstrate how we can use this analysis for a translation which combines two translations in a more complicated way then applying them serially on the source model.

### 5.2 Mixing translations steps

We have seen strongly faithful property transformations both for the translation which adds fail states to the model, and for the translation which does operation refinement (where the translation also adds a new state variable which allows distinguishing between the original states and the new, intermediate states). Now we want to find a property transformation for a translation which does both changes to the model. The changes though, are not done serially, one after the other, but together, so that it seems as if we can’t use the analysis we did for a serial composition of transitions, at least not directly.

First, however, we have to define how the translation combines the two translations, e.g., of fail state addition, and refinement. In the fail addition translation, we assumed that from a certain state, the translated system has more transitions than the original one, because the translated system ‘guesses’ the next state. Some of these ‘guesses’ prove to be wrong (leading to states which do not exist in the original model) and then these states are marked as fail states. We assumed that all the guessing and marking is done in one step, and when we make a wrong guess, the system recognizes this state immediately and it is marked as a fail state. But a more realistic scenario is that the guessing process is done in a few steps with inner calculations and decisions. Only then, after a few intermediate stages, does the system check if the state reached is legal (one of the sons in the original model), and mark it as a fail state or not, accordingly.

In this new scenario, we assume that the system passes through a few intermediate states between the states of the original model, and the next states could either be also states of the original model, or fail states. So like in the case of the refinement translation, we can assume that the system has some sort of a ‘Visible’ flag, which is turned on in the intermediate states, and then turned
back off.

From here we can draw conclusions on how the result model looks in comparison to the origin model, and in comparison to applying each of the two translations alone. Like in the refinement translation, from the states of the original model (which are marked as 'Visible' states) the system passes to intermediate states marked as not 'Visible'. The non-visible states, also like in the refinement translation, can be organized in any complicated structure, with non-deterministic decisions and sharing of states between different paths. But what we do know is that after a finite number of intermediate states, the path should reach a visible state, which can be either one of the sons of the state which we started from in the original model, or it can be a state marked as fail (in case this turns out to be a 'wrong guess'). In addition because we know that the system doesn't 'lose' any of the original sons in the guessing process, then it must be that for every son of the original state, we can reach it from the state, after a finite series of non-visible states. Moreover, as in the usual scenario of fail state addition, the system cannot 'recover' from a fail state: after it reaches a fail state, all the states from then on will also be marked as 'Fail'. This description defines a new translation, $T_R$, which is a combination between the translations of refinement, and fail-state addition.

5.2.1 The property transformation

As we saw before, we can get a property transformation by a serial composition of two other property transformations. The resulting transformation depends on the order of the composition. We also saw that, in certain conditions, if the two transformations are strongly faithful for two translations, then their composed transformation is strongly faithful for a translation which is a serial composition of the two translations, in the same order of composition as that of the transformations.

In this case, the translation discussed is not a serial composition of the two translations in any order. As it was described, it does not work on the model in two stages, first applying one translation and then the other one. It does not pass through an intermediate stage in which one of the translations was already completely applied, and the other one still not.

Still, as we will show below, if we indicate by $tr_1$ the transformation for the fail addition translation, and by $tr_2$ the transformation for the refinement translation, then the composed transformation $tr_2-tr_1$ is strongly faithful for the translation discussed here.

5.2.2 Relations between translations

To prove this we use a more general claim about a relation between two translations.

Definition - For two translations $TR_1$ and $TR_2$, we say that $TR_2$ is 'smaller' than $TR_1$, if for every two models $M_1$ and $M_2$, $TR_2(M_1,M_2) \Rightarrow TR_1(M_1,M_2)$
(so $TR_1$ relates all the model pairs which $TR_2$ relates, and maybe also additional ones).

Claim 1 - For two translations $TR_1$ and $TR_2$, such that $TR_2$ is a smaller translation than $TR_1$, if $tr$ is a strongly faithful property transformation for $TR_1$ then it is also a strongly faithful property transformation for $TR_2$.

Proof - for every two models $M_1$ and $M_2$, if $TR_2(M_1,M_2)$ then it is also true that $TR_1(M_1,M_2)$ and so because $tr$ is strongly faithful for $TR_1$ then $M_1 \models \phi \Rightarrow M_2 \models tr(\phi)$ and so it is also true that $tr$ is strongly faithful for $TR_2$.

Claim 2 - Assume we have two translations $TR_1$ and $TR_2$, with strongly faithful transformations $tr_1$ and $tr_2$, and a third translation $TR$. If for every two models $M_1$ and $M_3$ such that $TR(M_1,M_3)$ there exists a model $M_2$ such that $TR_1(M_1,M_2)$ and $TR_2(M_2,M_3)$ then the composed transformation $tr_2-tr_1$ is strongly faithful for the translation $TR$.

Proof - $TR$ is a smaller translation than the composed translation $TR_2-TR_1$ because for every two models $M_1$ and $M_3$ such that $TR(M_1,M_3) \Rightarrow \exists M_2$ such that $TR_1(M_1,M_2) \land TR_2(M_2,M_3) \Rightarrow TR_2-TR_1(M_1,M_3)$. We saw before that $tr_2-tr_1$ is strongly faithful for the translation $TR_2-TR_1$ and so by claim 1, it is also strongly faithful for $TR$.

5.2.3 Back to the property transformation

Let us indicate by $TR_1$ the fail-addition translation, by $TR_2$ the refinement translation, and by $TR$ the translation discussed here, which does both things together. We will prove that for any $M_1$ and $M_3$ which satisfy $TR(M_1,M_3)$, there exists a model $M_2$ such that $TR_1(M_1,M_2)$, and $TR_2(M_2,M_3)$.

Definition of $M_2$ - $M_2$ will have all the state variables of $M_1$, and in addition, the 'Fail' flag, but it will not have the 'Visible' flag. $M_2$ will contain all the visible states from $M_3$, both the fail, and the non-fail ones. For every two states in $M_2$, $s_1$ and $s_2$, there will be an edge from $s_1$ to $s_2$ iff in $M_3$ there exists a path from $s_1$ to $s_2$ which passes only through intermediate states which are non-visible states.

Proof that $TR_1(M_1,M_2)$ - We need to show that $M_2$ can be obtained from $M_1$ by marking all the original states as non-fail states, and adding sons which are fail states, and which continue only to other fail states. First, all the states of $M_1$ appear in $M_2$ and they are marked as non-fail states, because by the definition of our combined translation $TR$, they all appear in $M_3$ marked as non-fail states. Every non-fail state has all its original sons from $M_1$, because again, by the definition of $TR$, they are all connected to it by a series of non-visible states in $M_3$, and so are its sons in $M_2$ (by the way we derived $M_2$ from $M_3$). Also by the definition of $TR$, the only states which are connected to a visible state in $M_3$ by a series of non-visible states, are either its sons from $M_1$, or fail states. Thus in $M_2$, all the sons of a state are either its sons from $M_1$, or fail states. The last thing left to show is that all the sons of a fail state in $M_2$ are also fail states. This is true, because in $M_3$ (because of the definition of $TR$) all the descendants of a fail state are also fail states, and by the way we derived $M_2$ from $M_3$ it is true that all the sons of a state in $M_2$ are descendants.
of that state in $M_3$, so they must be fail states.

**Proof that $TR_1(M_1, M_2)$**: We need to prove that $M_3$ can be obtained from $M_2$, by marking all the states in $M_2$ as visible, and adding non-visible states between them. This is a direct consequence of the way we derived $M_2$ from $M_3$, by eliminating all the non-visible states, and replacing all the series of them with direct edges.

**Conclusion**: We showed that for every two models $M_1$ and $M_3$ such that $TR(M_1, M_3)$ there exists a model $M_2$ such that $TR_1(M_1, M_2)$ and $TR_2(M_2, M_3)$. Thus by claim $2$, the composed transformation $tr_2-tr_1$ is strongly faithful for the translation $TR$.

**References**


