Compiler correctness and the translation of logics
Janssen, T.M.V.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
1 Introduction

The first subject one thinks of, when one hears ‘translating’, probably is translating from one natural language to another (e.g. Chinese into English). Some readers might next think about translations between logics. But there are many more context where translating occurs. Frequent use of translations is made in computers: many different languages are used, and automatic translating between such languages is daily practice. For instance, the user describes the information he needs in some for him suitable view language, and the computer translates it into a data base query. Or the user describes his instructions to the computer in some programming language, and the compiler translates that into machine instructions. Translations in different fields are compared by Janssen (1998) and steps are made to a general theory of translating. In this article the issue of compiler correctness will be investigated, and the insights obtained there, are applied to the translation between logics.

2 Correctness of compilers

In order to steer a computer, languages are designed which reflect closely the structure of a computer: they speak, for instance, about memory cells and stacks with values. They are called machine languages or assembly languages. However, when some scientist or administrator wants to express his instructions to the computer, these languages are not so easy to handle. Therefore soon after the first computers were built, languages have been developed that allow humans to express there instructions more easily. One of the first was FORTRAN, which still is in use; JAVA is a modern one. The compiler is the computer program that translates from a (higher level) programming language into a (low level) assembly language. Since one wants to be sure that the computer performs as intended, correctness of the translation is a crucial issue.

How to characterize correctness formally, how to organize the translation process in such a way that correctness could be proven? We will consider an approach that became one of the most important approaches in the theory of compiler construction.

The source article of the approach was ‘Advice on structuring compilers and proving them correct’ (Morris 1973). The programming language and the assembly are interpreted in formal models. For the assembly language that is a model which can be seen as an idealized computer, with memory cells and values stored in such cells. Then meanings can be, for instance, state changing functions in such a model. The programming language is interpreted
a more abstract model; and what meanings are there, depends on what the
language speaks about. Part of the advice was to design the languages and
semantic models as algebras, and to define the translations and interpretations
as homomorphisms. The translation can then be defined by giving the translations
of the generators in the algebra for the source language into polynomials over
the target language algebra. The same translation method is known in the
field of translations of logic under other names: the grammatical translation
(Epstein 1995) or the schematic mappings (Feitosa & d’Ottaviano 2001). The
crux of the proposal was to define correctness as commutativity of the leftmost
diagram in Fig. 1, which then could be proven by finite algebraic means.

Several variants of Morris proposal occur in the literature. Some introduce
algebraic tools in order to facilitate the aims, others have a different correctness
definition. We discuss one alternative, for other variants, see Janssen (1998).

Thatcher et al. (1979) use more algebra than Morris (1973), and they propose
to define compiler correctness as commutativity of the rightmost diagram (in
which the lower arrow points from left to right). This proposal could
not be their last word on the issue, because a counterexample can easily be
given. Suppose the programming languages has a notation for both positive
and negative numbers, but the assembly language has no information on signs.
Then in the computer model there is only one kind of numbers, say only positive
numbers. There is a encoding homomorphism: assign to +3 and −3 the same
number, viz. 3. This of course not what intuitively would be called a correct
compiler because essential information from the programming language gets lost
in translation. As matter of fact, Thatcher et al. admit that their definition is
not fully adequate. They say [names and ref. adapted TJ]: ‘[...] commuting of
the rightmost diagram in Fig. 1 is not, in itself “compiler correctness”. ProgL
and AssL could be one point algebras and Compile, Int PL and encode the
unique homomorphisms to those one point algebras resulting in a commutative
square. One possibility around this degenerative case [...] would be to require
the encoding to be injective (it is in our case) and that condition is certainly
sufficient. We are just not sure at this time that it is necessary.’

That is an attractive proposal: if the encoding is an injective homomorphism,
the encoding becomes an isomorphism, and then the translation exactly mimics

\[
\begin{array}{ccc}
\text{PrL} & \xrightarrow{\text{Compile}} & \text{AssL} \\
\text{Int}_{\text{PL}} & & \text{Int}_{\text{AL}} \\
\text{Sem(PrL)} & \xrightarrow{\text{Decode}} & \text{CompMod} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{PrL} & \xrightarrow{\text{Compile}} & \text{AssL} \\
\text{Int}_{\text{PL}} & & \text{Int}_{\text{AL}} \\
\text{Sem(PrL)} & \xrightarrow{\text{Encode}} & \text{CompMod} \\
\end{array}
\]

Figure 1: Compiler correctness.
Morris (1973): correctness is commutativity of the leftmost diagram,
Thatcher et al. (1979): correctness is commutativity of the rightmost diagram.
PrL: Programming Language \hspace{1cm} Int_{PL}: Interpretation of PrL
AssL: Assembly Language \hspace{1cm} Int_{AL}: Interpretation of AssL
Sem(PrL): meanings of the PrL \hspace{1cm} CompMod: abstract model of computer
the intended meaning of the source language. However, the differences between
the programming language and assembly language can be considerable, and
only in exceptional cases isomorphy arises. We will consider an example that
illustrates this, and helps to appreciate the correctness definition; first it is
described abstractly, next with historical details. The example will also be
relevant for the case of translations between logics.

Suppose the programming language deals with numbers, and has two syn-
tactically distinct expressions for the number zero, viz. +0 and −0. Suppose
moreover that these are translated in different expressions in the assembly lan-
guage and that their interpretation in the machine model differs as well. So
at the bottom of the diagram there cannot be a function from left (one num-
ber zero) to right (two representations in the computer model), and there is no
isomorphy: there only is a function from right to left.

This example was not designed for the purpose of defending a correctness
definition; it describes a real situation. Computers with such a number representa-
tion systems were made in the seventies, an example was the CDC computer.
Numbers were represented in a binary format, and a initial sign bit indicated
whether it was positive or negative; then 00000 was +0, and 10000 was −0,
00001 was +1, 11110 was −1. The advantage was that changing the sign of a
number was very easy: changing all 0’s into 1’s. The disadvantage was that
arithmetical operations might yield +0 or −0 depending on operands. Scien-
tists did not appreciate that situation. In computers from a later period (e.g.
the IBM360) each number had its own representation as a string, and zero was
always encoded as a positive number. Attractive, but there also was a disad-
vantge. Since there is an even number of binary strings of fixed length, the
computer could either not represent as many negative numbers as positive ones,
or one bit pattern was left over. For an extensive discussion of these representa-
tion methods, see Tanenbaum (1976); in the third edition (Tanenbaum 1999,
p. 559) he states that the first method is obsolete now.

Also general considerations can explain this situation. Let some program-
matic language be given together with its intended meaning. Since the compiled
program should do what it has to do according to the semantics of the pro-
gramming language, going through a compiler should be a way to obtain the
originally intended semantics. Hence the meanings of the assembly language
should be interpreted in the intended semantics in order to see whether the
compiler yields the intended results. So compiler correctness consist in the ex-
istence of a decoding mapping such that the left diagram in Fig. 1 commutes.

3 Application to logic

An application of the ideas from the previous sections to logic is not straight-
forward because it is not always easy to say what the meaning of a logic is. If a
logic is sound and strongly complete with respect to a certain class of models one
may consider the class of models in which the a formula is true as its meaning,
and if both involved languages are of this nature, one may have, as counterpart
of the translation, a transformation that changes models. Sometimes this is the
case: Epstein (1995, p. 394) presents a translation from Intuitionistic Logic (Int)
into a modal logic (S4) together with a transformation on the models. However,
most logics are defined as a deduction system and then this approach is not
easily applied. Therefore Janssen (1998) hardly gives attention to translations of logics.

Another approach is possible. As a general approach to translations Carnielli & d’Ottaviano (1997) and Feitosa & d’Ottaviano (2001) have applied the view that logics are languages with a consequence relation, and judge mappings for being a translation on the basis of this consequence relation. The relevant definitions from their papers are given below, but with an adopted terminology. I prefer to use ‘translation’ as a general term for the map between languages, and call ‘consequence preserving translation’ what they call ‘translation’.

**Definition 3.1.** A *logic* is a pair \( L = (L, C_L) \), where \( L \) is a formal language and \( C_L \) a consequence operator, that is a function \( C_L : \wp(L) \rightarrow \wp(L) \) that satisfies for \( X, Y \subseteq L \), the following conditions:

1. \( X \subseteq C_L(X) \),
2. If \( X \subseteq Y \), then \( C_L(X) \subseteq C_L(Y) \),
3. \( C_L(C_L(X)) \subseteq C_L(X) \)

**Definition 3.2.** A *consequence preserving translation* from logic \( K = (K, C_K) \) into logic \( L = (L, C_L) \) is a translation \( T : K \rightarrow L \) such that, for every \( X \subseteq K \):

\[
T(C_K(X)) \subseteq C_L(T(X)).
\]

The idea that the meaning of a formula is the set of its consequences is an attractive aspect because this can be applied to a semantically defined consequence relation as well as to a syntactically defined one. When we represent the above definition of consequence preserving translation in the format of the previous section, we obtain the rightmost diagram in Fig. 2 as the representation for what in the just mentioned papers is called a translation. Note that the structure of the expressions in the logic plays is not reflected in the consequence relation, so the algebraic aspects play no role in the diagram.

**Figure 2:** Translations between logics: Commutative diagrams

Right: Translation according to Feitosa & d’Ottaviano (2001)

Left: Alternative proposed here.

\( K \): language of a logic \( K = (K, C_K) \).
\( C_K \) consequence relation of \( K \).

\( C_K(K) = \{ C_K(A) \mid A \in K \} \), i.e. the set of consequence sets.

Analogously for \( L \) and \( C_L \).

*Preserve* is *Translate* lifted to sets of consequences, *Respect* for *Translate* \(^{-1}\).

This diagram is analogous to the rightmost diagram in Fig. 1. Warned by that discussion one might worry about degenerated translations. Indeed, if we translate *all* expressions from the source language into one and the same
expression of the target language, the translation is consequence preserving. So any logic can be translated consequence preserving into any logic. Another trivial case is that any logic can be translated consequence preserving into an inconsistent logic. So Def. 3.2 on its own does not seem to capture an interesting notion.

My proposal is to follow the idea from the previous section and to define a restriction on the mapping from right to left, so in the other direction than in the definition above, see the leftmost diagram in Fig. 1. Instead of calling it a correct translation (as for compilers), it seems, in the present context, more appropriate to speak about a consequence respecting translation.

Definition 3.3. A consequence respecting translation from logic $\mathcal{K} = \langle K, C_K \rangle$ into logic $\mathcal{L} = \langle L, C_L \rangle$ is a map $T: K \rightarrow L$ such that for any $A \in K$ holds:

\[
\text{if } B \in C_L(T(A)) \text{ then } T^{-1}(B) \subseteq C_K(A).
\]

This type of translation has an interesting consequence.

Definition 3.4. A logic $\mathcal{L}$ is consistent iff for some $B \in L$: $B \notin C_L(\emptyset)$.

Theorem 3.5. If $T: K \rightarrow L$ is a consequence respecting translation, and $K$ is consistent, then $L$ is consistent.

Proof. Assume $K$ to be consistent, and assume $L$ to be inconsistent. Let $B \notin C_K(\emptyset)$. Since $L$ is inconsistent, $T(B) \in C_L(\emptyset)$, so $T^{-1}(T(B)) \subseteq C_K(\emptyset)$, hence $B \in C_K(\emptyset)$. Contradiction. So $L$ must be consistent.

As a special case we may consider the conservative translations. The definition below is the same as the one by Carnielli & d’Ottaviano (1997) and Feitosa & d’Ottaviano (2001); they give examples and investigate category theoretic properties of conservative translations. It is an attractive form of translation, but as was the case for compilers, it is a strong condition that in several interesting cases cannot be achieved, see Sect. 4. Less restrictive is the notion faithful translation, a type of translation that often is used in the literature.

Definition 3.6. A conservative translation from $\mathcal{K}$ into $\mathcal{L}$ is a function $T: K \rightarrow L$ such that for every set $X \subseteq K$ and $A \in X$:

\[
A \in C_K(X) \iff T(A) \in C_L(T(X)).
\]

Definition 3.7. A faithful translation from $\mathcal{K}$ into $\mathcal{L}$ is a function $T: K \rightarrow L$ such that for every $A \in K$: $A \in C_K(\emptyset) \iff T(A) \in C_L(\emptyset)$.

Theorem 3.8. A translation $T$ is conservative if and only if it is consequence preserving and consequence respecting.

4 Examples

We will illustrate our definition of consequence respecting translations by some examples.
4.1 Translation of $\text{Int}$ into $\text{PL}$

We consider intuitionistic propositional logic ($\text{Int}$). In that logic a proposition $A$ is interpreted as ‘I have a proof for $A$’, $\neg A$ as ‘I do not have a proof for $A$’, so from $\neg\neg A$ (‘I do not have a proof that I do not have a proof for $A$’) does not follow $A$ (‘I have a proof for $A$’). But from $A$ does follow $\neg\neg A$.

Let the translation $T$ from intuitionistic logic ($\text{Int}$) into propositional logic, defined by $T(A) = A$ for any $A \in \text{Int}$ (the identity map). One easily sees that this translation is consequence preserving. But, for any proposition letter $p$, $C_{\text{PL}}(T(p)) = C_{\text{PL}}(T(\neg\neg p))$. So the translations of two formulas which have in $\text{Int}$ distinct consequences are by $C_{\text{PL}}$ assigned the same sets of consequences. This means that there is no map from right to left, the translation is not consequence respecting.

Surprisingly, this translation is given as a motivation for the definition of consequence preserving translation (Carnielli & d’Ottaviano 1997, p. 72). ‘In the literature, definitions of translations between logics require, in general, that the converse of the condition [in Def. 3.2] also holds. We prefer the notion as defined in order to accommodate certain maps that seem to us as obvious examples of translations, such as the identity map from intuitionistic into classical logic’. For me this example counts as a translation because it is a map between the two languages, but otherwise I would neglect it, because it is a translation in which essential information about the intuitionistic meaning is lost.

4.2 Translating of $\text{PL}$ into $\text{Int}$

The Gödel interpretation $Gd$ from classical propositional logic $\text{PL}$ into propositional intuitionistic logic $\text{Int}$ is defined as follows:

- $Gd(p) = p$, for proposition letters
- $Gd(\neg A) = \neg Gd(A)$
- $Gd(A \land B) = Gd(A) \land Gd(B)$,
- $Gd(A \lor B) = \neg(\neg Gd(A) \land \neg Gd(B))$
- $Gd(A \rightarrow B) = \neg(Gd(A) \land \neg Gd(B))$.

Consider now the PL formulas $p$ (i.e. a simple proposition letter) and $\neg\neg p$. These formulas from $\text{PL}$ are translated into the formulas $p$ and $\neg\neg p$ of $\text{Int}$. In $\text{PL}$ these formulas are equivalent, so $C_{\text{PL}}(p) = C_{\text{PL}}(\neg\neg p)$. In intuitionistic logic finer distinctions are made and the meanings of $p$ and $\neg\neg p$ differ; in fact $C_{\text{Int}}(\neg\neg p) \subset C_{\text{Int}}(p)$. In spite of this distinction, the translation is a good encoding of the original consequence relation. We know what has to be added to the target language consequences in order to obtain those of the source meanings: add the axiom of the excluded third, and recalculate the consequences again. Hence $Gd$ is a consequence respecting translation.

In Feitosa & d’Ottaviano (2001) this Gödel interpretation is given as an example of a mapping that is not a translation according to their definition: $p \in C_{\text{PL}}(\neg\neg p)$ whereas $T(p) \notin C_{\text{Int}}(T(\neg\neg p))$, and therefore this translation does not preserve derivability. I welcome this as a translation because it respects derivability: it is consequence preserving. Moreover, it is faithful.
4.3 Translation of PL into Kleene’s three valued logic $K_3$

Kleene (1952) introduced a three-valued logic as a way to reason with propositions whose truth-value we do not or cannot know. Besides the truth values $T$ and $F$, he introduced the value $U$ (undefined). The truth tables are as follows:

<table>
<thead>
<tr>
<th>$A \land B$</th>
<th>$B$</th>
<th>$A \lor B$</th>
<th>$B$</th>
<th>$A \rightarrow B$</th>
<th>$B$</th>
<th>$A \neg A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$: $T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$F$</td>
<td>$T$</td>
</tr>
<tr>
<td>$U$</td>
<td>$U$</td>
<td>$T$</td>
<td>$T$</td>
<td>$U$</td>
<td>$T$</td>
<td>$U$</td>
</tr>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

Figure 3: Truth tables for Kleene’s three valued logic $K_3$

The tautologies for $K_3$ are defined as the formulas which for all valuations yield $T$. Since a formula with proposition letters which all take value $U$ is evaluated as $U$, the logic $K_3$ has no tautologies.

The map $\text{Id}: PL \rightarrow K_3$ which assigns to each formula the identical formula in $K_3$, is a information preserving translation from PL into $K_3$, as one sees as follows. The part of the tables of $K_3$ which deals with $T$ and $F$ is identical with the tables for propositional logic PL. So if $A$ is a consequence of $B$ in $K_3$, then $A$ is a consequence of $B$ in PL. The difference between the logics is that $C_{PL}(A)$ contains more consequences then $C_{K_3}(A)$, e.g. tautologies. In order to find $T^{-1}(C_{K_3}(A))$ we have, intuitively speaking, to add all tautologies and what can be obtained by using them. Or formulated more simply, apply $C_{PL}$ because $C_{PL}(\text{Id}^{-1}(C_{K_3}(\text{Id}(A)))) = C_{PL}(A)$. We may say that $K_3$ respects derivability of PL, it keeps the distinctions, but does not follow it exactly: it does not express the influence of tautologies.

As we noticed before, the logic has no tautologies, its theory is empty. Proposition 1.28 of Feitosa & d’Ottaviano (2001) states that there is no translation in their sense (consequence preserving) from a logic with a non-empty theory into one with an empty theory. One can easily see that for this case: tautologies that do not have an occurrence of $A$ are in $C_{PL}(A)$, but not in $C_{K_3}(T(A))$. It seems to me to be a useful translation; it is a faithful, but not a conservative one.

One sees the analogy with the previous case; the system $K_3$ makes more distinctions than PL, it discriminates e.g. between consequences that need the given premise, and those that do not.

4.4 Translation of PL into the paraconsistent logic $J_3$

A paraconsistent logic is a logic in which a non trivial theory may include both a proposition and its negation. A survey of the paraconsistent logic $J_3$ is given in chapter 3 of Epstein (1995)(written in collaboration with $J_3$’s creator d’Ottaviano).

In $J_3$ there are three truth values: 1 for truth, and $\frac{1}{2}$ for a degree of truth, and 0 for false. By definition $\models A$ holds iff $A$ has value 1 or $\frac{1}{2}$. The truth tables we need in our discussion are the ones for $\land$, $\rightarrow$, $\neg$, where $\neg A$ is a weak negation), and $\odot$, where $\odot A$ asserts that $A$ has a definite truth value (viz. 1 or 0), see Fig. 4.
Proof. We know that \( V_q \) respectively let \( guage \) can express. The other three examples illustrate that in case the target possible if the source language makes more distinctions than the target lan-

The first example illustrates that an information preserving translation is faithful, but not conservative.

So again we are in the situation that the target language does have fewer consequences than the source language, but keeps the distinctions from the source language. Note that the translation is faithful, but not conservative.

**The translation \( \ast \): PL \( \rightarrow J_3 \) is not consequence preserving.**

Proof. We know that \( p \land \neg p \models_{PL} q \). Furthermore \( (p \land \neg p)^* \) \( = (p \lor q) \) and \( q' = (p \land q) \). We show that \( (p \lor q) \not\models_{J_3} (p \land q) \). Let \( V(p) = \frac{1}{2} \) and \( V(q) = 0 \). Then \( p \) has no definite truth value, so \( V(p) = 0 \), hence \( V((p \lor q) \not\models_{J_3} (p \land q)) = 1 \). But \( V((p \lor q) = 1) \), so \( V((p \land q) = 0) \).

\( T \) respects the consequence relations of PL: they are not mixed up and can be reconstructed from the consequences in of the translation. One easily checks that \( T(p \land \neg p) \models_{J_3} T(p) \). So \( J_3 \) distinguishes between consequences based upon contradiction and those based upon conjunction reduction. So again we are in the situation that the target language does have fewer consequences than the source language, but keeps the distinctions from the source language. Note that the translation is faithful, but not conservative.

5 The translation paradox

The first example illustrates that an information preserving translation is not possible if the source language makes more distinctions than the target language can express. The other three examples illustrate that in case the target
logic makes more distinctions than the source logic, a information preserving translation may be possible.

Béziau (1999) introduced $PL/2$, a variant of $PL$ in which only one half of the meaning of the negation is given: if $V(\varphi) = T$, then $V(\neg \varphi) = F$. He showed that $PL$ can faithfully be translated into $PL/2$, by $T(\neg A) = T(A) \rightarrow \neg T(A)$, and $T(A) = A$ in the other cases. So it is an example where a stronger logic is faithfully translated into a weaker logic that is included in it. Next he considers another example: the Gödel translation (discussed here in Sect. 4.2). He says: ‘the fact that classical logic can be translated into intuitionistic logic, which is strictly included into it, is still a paradox because it is against intuition and has not yet been properly explained’. Three of our examples are of this nature.

Our perspective on translating gives an explanation. If a logic is translated into a weaker logic, that logic has fewer rules, fewer formulas will be equivalent, more distinctions are made because. So the weaker logic is more expressive. If the translation is consequence respecting, it means that a reconstruction is made in the weaker logic in which no information about the stronger logic is lost in translation, so for questions about the stronger logic all information remains available. It is to not surprising that a logic can be translated into a weaker one. Vice versa would be surprising, because the additional power of the stronger logic is likely to mix up the consequence structure of the weaker logic. For instance, the map $Id: K_3 \rightarrow PL$ would not work because the non equivalent $K_3$ formulas $A$ and $A \land (B \lor \neg B)$ are translated into formulas which have in $PL$ the same consequences. This would not be a consequence respecting translation.

Humberstone (2005) discusses Béziau's example concerning $PL/2$, and mentions the explanation we just gave. He claims that ‘this response fails’ and gives an example of a modal logic that cannot be translated into the weakest modal logic $L$. However, I am not convinced by his example. First, he requires the translation to be a ‘definitional translation’, that is a translation in which a propositional variable is translated into itself. Hence a complex translation as in our $K_3$ example ($T(p) = \circ p \rightarrow p$, encoding that $p$ has a definite truth value) is not allowed. So he disallows that a different perspective on basic propositions is encoded by the translation. Secondly, he requires the translation to be a conservative one (in his terminology ‘faithful embedding’). So a translation must not only encode the source logic, but give an isomorphic reconstruction. Combining all this, is difficult, and sometimes not possible.

6 Conclusion

We have shown that the insights from compiler theory can be transferred to the theory of translating logics, provided we conceive a logic as a language with a consequence relation. Inspired by compiler theory, we found the definition ‘consequence respecting translation’ which was defined as commutativity of a certain diagram. The alternative definition (consequence preserving) was shown to suffer from the same shortcomings as the corresponding correctness definition in compiler theory. Some examples illustrated the attractiveness of our notion of translation. The insights obtained from compiler theory explain that Béziau’s translation paradox in fact describes a phenomenon that is to be expected.
References


Tanenbaum, A. S. (1976), Structured computer organization, Prentice-Hall.
