Translation from Set-Theory to Predicate Calculus

Matthias Konrad (ETH Zurich) Laurent Voisin (Systerel)

1 March 2012

Contents

The mathematical language event-B contains powerful syntactic constructs to reason about functions, relations and some other sets. Automatic provers on the other side prefer to work on the smallest syntactic subset of the language that is still expressive.

This document describes a translation which:

- removes most set-theoretic constructs of a predicate.
- separates arithmetic and set-theoretic constructs from each other.
- simplifies predicates.

Revision History

Date	Contents
2012/03/01	Fixed rule ER10 which was incorrect (allowed negative cardinal numbers).
2012/02/23	Revised translation of expressions bearing a Cartesian product type by introducing new rule IR3'.
2011/07/08	Removed $\neq$ from the target language (it was not reachable). Added an option mechanism for tailoring the translation. Currently supported options are `expandSetEquality` and `aggressiveSimplification`.
2010/03/03	Added rules IR47 and IR48 for translating $\prec$ and $\succ$.
2009/04/17	Updated to mathematical language V2.
2007/11/23	Added extraction of bool operator in rule ER13.
2007/08/23	Strengthened IR3 so that it's applied only when the right-hand side is an identifier. Removed the conditional quantification from IR34.
2007/07/20	Replaced rule IR22 by rule IR2'. Changed rule ER9 to expand set equality when one of the two sets is a type. Simplified rule ER9 to use equivalence rather than double inclusion.
2006/05/16	Initial revision

1 Abstract Syntax

The goal of this transformation is a predicate expression, which is almost free of set-theoretic elements. The following abstract syntax defines this goal.
The production rules for predicates are:

$$\begin{aligned}
\text{pred-bin: } & P ::= P_1\ P_2\ [\textit{pred-binop}] \\
\text{pred-ass: } & P ::= P_1\ P_2\ \ldots\ P_n\ [\textit{pred-assop}] \\
\text{pred-una: } & P ::= P_1 \\
\text{pred-quant: } & P ::= L_1\ P_1\ [\textit{pred-quant}] \\
\text{pred-lit: } & P ::= [\textit{pred-lit}] \\
\text{pred-rel: } & P ::= AE_1\ AE_2\ [\textit{pred-relop}] \\
\text{pred-in: } & P ::= ME_1\ SE_1 \\
\text{pred-setequ: } & P ::= SE_1\ SE_2 \\
\text{pred-boolequ: } & P ::= BE_1\ BE_2 \\
\text{pred-identequ: } & P ::= I_1\ I_2
\end{aligned}$$

where

$$\begin{aligned}
&\textit{pred-binop} \in \{\text{limp}, \text{leqv}\} \\
&\textit{pred-assop} \in \{\text{land}, \text{lor}\} \\
&\textit{pred-quant} \in \{\text{forall}, \text{exists}\} \\
&\textit{pred-lit} \in \{\text{btrue}, \text{bfalse}\} \\
&\textit{pred-relop} \in \{\text{equal}, \text{lt}, \text{le}, \text{gt}, \text{ge}\} \\
&\text{type}\,(I_1) = \text{type}\,(I_2) = \text{carrier set}\ .
\end{aligned}$$

The production rules for lists of identifiers and identifiers are:

$$\begin{aligned}
\text{ident-list: } & L ::= I_1\ I_2\ \ldots\ I_n \\
\text{ident: } & I\ ::= [\textit{name}]
\end{aligned}$$

where

$$\begin{aligned}
&1 \leq n \\
&\textit{name} \text{ is a string of characters.}
\end{aligned}$$

The production rules for arithmetic expressions are:

$$\begin{aligned}
\text{a-expr-bin: } & AE ::= AE_1\ AE_2\ [\textit{a-expr-binop}] \\
\text{a-expr-ass: } & AE ::= AE_1\ AE_2\ \ldots\ AE_n\ [\textit{a-expr-assop}] \\
\text{a-expr-una: } & AE ::= AE_1\ [\textit{a-expr-unop}] \\
\text{a-expr-ident: } & AE ::= I_1 \\
\text{a-expr-int: } & AE ::= [\textit{int-lit}]
\end{aligned}$$

where

$$\begin{aligned}
&\textit{a-expr-binop} \in \{\ \text{minus}, \text{div}, \text{mod}, \text{expn}\ \} \\
&\textit{a-expr-assnop} \in \{\ \text{plus}, \text{mul}\ \} \\
&\textit{a-expr-unop} \in \{\ \text{uminus}\ \} \\
&\textit{int-lit} \text{ is an integer number.} \\
&\text{type}\,(I_1) = \mathbb{Z}
\end{aligned}$$

The production rules for boolean expressions:

$$\begin{aligned}
\text{b-expr-atom: } & BE ::= [\textit{b-expr-lit}] \\
\text{b-expr-ident: } & BE ::= I_1
\end{aligned}$$

where

$$\begin{array}{l} \textit{b-expr-lit} \in \{\ \text{true}\ \} \\ \text{type}\,(I_1) = \text{BOOL} \end{array}$$

The production rules for maplet expressions are:

$$\begin{array}{rl} \textsf{m-expr-maplet}\text{:} & ME ::= ME_1\ ME_2 \\ \textsf{m-expr-ident}\text{:} & ME ::= I_1 \\ \textsf{m-expr-atom}\text{:} & ME ::= [\textit{m-expr-lit}] \end{array}$$

where

$$\begin{array}{l} \textit{m-expr-lit} \in \{\text{integer}, \text{bool}\} \\ \text{type}\,(I_1) \neq \alpha \times \beta \end{array}$$

The production rules for set expressions are:

$$\textsf{s-expr-ident}\text{:}\ SE ::= I_1$$

where

$$\text{type}\,(I_1) = \mathbb{P}\,(S)\ \text{ and } S \neq I_1$$

2 Maplet Decomposition

The translation rules described in the following sections suppose that all expressions of Cartesian product type have been decomposed so that they take only the form of scalar expressions separated by maplets.

Having a close look at the specification of type checking of event-B mathematical formulas, it happears that very few expressions can bear a Cartesian product type, namely:

$$\begin{array}{l} \textsf{expr-ident} \\ \textsf{expr-bin}\ [\textit{funimage}] \\ \textsf{expr-bin}\ [\textit{mapsto}] \end{array}$$

We therefore have to take special care with identifiers and functional images.

2.1 Identifier Decomposition

As concerns identifiers, we apply a pre-translation processing to decompose every bound and free identifier into several identifiers, so that no identifier bears a Cartesian product type when entering translation. After this pre-processing, all identifiers bear a type which is either set, boolean or integer.

The decomposition is done by introducing new bound identifiers. How and where they are introduced depends on whether the decomposed identifier is bound or free and will be looked at later.

The actual decomposition of an identifier is done according to the decomposition operator *dc*:

$$\text{dc}\,(i) = \begin{cases} \text{dc}\,(i_0) \mapsto \text{dc}\,(i_1) & \text{if } i \in S_1 \times S_2 \wedge i_0 \in S_1 \wedge i_2 \in S_2 \\ \text{new bound identifier} & \text{otherwise} \end{cases}$$

If a bound identifier is decomposed, its bound identifier declaration is replaced by:

$$\text{free}\left(\text{dc}\left(i\right)\right)$$

For instance, predicate

$$\forall x \cdot\ 10 \mapsto (20 \mapsto 30) = x$$

becomes

$$\forall x, x_0, x_1 \cdot\ 10 \mapsto (20 \mapsto 30) = x \mapsto (x_0 \mapsto x_1)$$

after identifier decomposition.

For the bound identifiers generated by the decomposition of the free identifiers $(f_1 \ldots f_n)$ of a predicate P, a new forall quantification is introduced around the original predicate. The identifier declaration of this quantification consists of the union of:

$$\text{free}\left(\text{dc}\left(f_1\right)\right), \text{free}\left(\text{dc}\left(f_2\right)\right), \ldots \text{free}\left(\text{dc}\left(f_n\right)\right)$$

The quantified predicate is of the form:

$$f_1 = \text{dc}\left(f_1\right) \wedge f_2 = \text{dc}\left(f_2\right) \wedge \ldots f_n = \text{dc}\left(f_n\right) \Rightarrow P'$$

where P' is the result of substituting every free identifier f_i by its decomposition $\text{dc}\left(f_i\right)$.

For instance, predicate

$$a = b \wedge a \in S$$

with typing

$$a \in S \wedge b \in S \wedge S \in \mathbb{P}\left(\mathbb{Z} \times \mathbb{Z}\right)$$

is transformed into

$$\begin{array}{ll} \forall x_0, x_1, x_2, x_3 \cdot & a = x_0 \mapsto x_1 \ \wedge\ b = x_2 \mapsto x_3 \\ & \Rightarrow \\ & x_0 \mapsto x_1 = x_2 \mapsto x_3 \ \wedge\ x_0 \mapsto x_1 \in S\ . \end{array}$$

2.2 Functional Image Decomposition

As concerns functional images, decomposition is applied on the fly during translation by some special rules, namely ER13 and IR3'.

3 Translation Rules

In this section several sets of rules are introduced. With these rules repeatedly applied, it should be possible, to bring any set theoretic expression into the form described in the former section.

As presented in the first section, a predicate can contain expressions of various forms:

- Predicate
- Arithmetic (either a number or something encapsulated in an arithmetic expression)

- Maplet (either an identifier or something encapsulated in a maplet expression)
- Set

For instance, the predicate P_1:

$$\max(\{x \cdot x > 2 \ \wedge \ x < 10 \mid x\}) > 2$$

is structured as follows:

1. P_1 encapsulates, through $>$, two arithmetic expressions $A_1 \mathrel{\widehat{=}} \max(\ldots)$ and $A_2 \mathrel{\widehat{=}} 2$.
2. A_1 encapsulates, through max, the set expression $S_1 \mathrel{\widehat{=}} \{x \cdot \ldots \mid x\}$.
3. S_1 encapsulates, through set comprehension, the predicate $P_2 \mathrel{\widehat{=}} x > 2 \ \wedge \ x < 10$.
4. P_2 encapsulates through $>$, $<$ other arithmetic expressions.

For the sake of translating the predicates, it is important to analyse these borders. And accordingly understand how and through which operators, the different formula forms can be encapsulated in each other.

This is done in figure 1. The arrow from Arithmetic to Set shows for example, that sets may be encapsulated in Arithmetic expressions by using either min, max, card or function application.

Figure 1 furthermore provides an overview about the overall translation process.

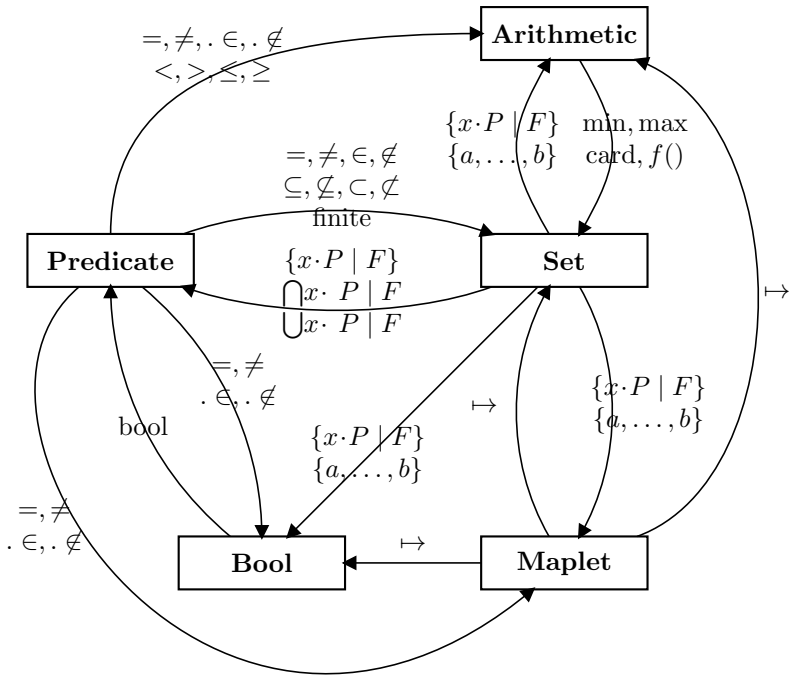

Figure 1: Borders of the initial formula structure

3.1 Basic Rules

The first set of rules presented, eliminate all borders for the operators:

$$\neq, \notin, \subseteq, \not\subseteq, \subset, \not\subset, \text{finite}$$

Apart from $\neq$, they all work on the border between predicate and set. These rules applied properly, the only way left to directly encapsulate a set in a predicate is through the operators $\in$ and $=$.

Name	**Mask**	**Right**	**Remark**
BR1	$s \subseteq t$	$s \in \mathbb{P}(t)$	
BR2	$s \not\subseteq t$	$\neg(s \in \mathbb{P}(t))$	
BR3	$s^2 \subset t^2$	$s \in \mathbb{P}(t) \ \wedge \ \neg(t \in \mathbb{P}(s))$	
BR4	$s^2 \not\subset t^2$	$\neg(s \in \mathbb{P}(t)) \ \vee \ t \in \mathbb{P}(s)$	
BR5	$x^2 \neq y^2$	$\neg(x = y)$	
BR6	$x \notin s$	$\neg(x \in s)$	
BR7	$\text{finite}(s^2)$	$\forall a \cdot \exists b, f \cdot \ f \in s'' \rightarrowtail a'..b$	(1)
BR8	$\text{partition}(s, s_1^n, s_2^n, \ldots, s_n^n)$	$s = s_1 \cup s_2 \cup \cdots \cup s_n$ $s_1 \cap s_2 = \varnothing$ $\vdots$ $s_1 \cap s_n = \varnothing$ $\vdots$ $s_{n-1} \cap s_n = \varnothing$	

Primed operands. In rule BR7, a new quantification is introduced around the existing expression s. Bound identifiers are implemented using the de Bruijn notation. Consider the following formula:

$$\forall x \cdot \ \text{finite}(\{x_0\})$$

The subscript 0 in the second occurrence of x denotes the fact that it corresponds to the nearest bound identifier declaration, when going to the left. When rule BR7 is applied to the predicate of the quantification, the following is generated (the indices for the de Bruijn notation are again showed as subscripts):

$$\forall x \cdot \ \forall a \cdot \ \exists b, f \cdot \ f_0 \in \{x_3\} \rightarrowtail a_2..b_1$$

The de Bruijn index of x changed from 0 to 3, because there are now three new bound identifier declarations between the usage of x and its declaration.

In the rule description of BR7, s is primed twice becoming s''. This denotes the fact, that the externally bound de Bruijn indices of s have to be incremented by the amount of identifiers introduced by the first two quantifications to the left.

Operands with superscripts. Some rules generate multiple occurrences of a given operand. Since an operand can itself be a huge subformula, this is a bad thing and may lead to formula explosion. Such operands are thus superscripted by a number, which states how many times they will occur in the final formula.

In rule BR3, s and t are super-scripted by 2, because they occur twice in the resulting formula. In rule BR7, s is also super-scripted by 2, even though s appears only once in the right-hand side. The superscript indicates that the membership rule for a total injection will duplicate s in a later stage.

The superscripts for BR5 are only valid if x and y are of a set type. Superscripts should thus be interpreted as a worst case measure.

3.2 Comparison Rules

Even though they are presented together, the different equality rules do stuff quite unrelated. Some explanations are thus needed. It is assumed, that the basic rules where already applied.

ER2 Removes direct $=$ encapsulations of maplets in predicates. This rule and the basic rules applied, the only way left to directly encapsulate a maplet in a predicate is through the $\in$ operator.

ER3-ER7 Remove all direct encapsulations of boolean expressions in predicates that go through the $=$ operator. The only way left to directly encapsulate a boolean expression in a predicate is through the $\in$ operator.

ER8, ER9 Remove all direct encapsulations of sets in predicates through the operator $=$. The only way left to directly encapsulate a set in a predicate is through the $\in$ operator.

ER8, ER10-ER12, CR1-CR5 Remove comparison constructs but only in very specific situations. In most cases, these rules are only effective after a predicate is reorganized.

ER13, CR6, CR7 Extract set constructs (either min, max, card or function application) from arithmetic expressions and bool operators from boolean expressions. After these rules have been applied, the predicate will be in the right form to be reduced by the rules ER8, ER10-ER12.

Conditional quantification. Some rules use conditional quantification, i.e., introduce a new quantification when some operands are not proper maplets. A conditional quantification is denoted as:

$$\forall_c\ P(e_1, e_2^*, \ldots, e_n) \quad \text{or} \quad \exists_c\ P(e_1, e_2^*, \ldots, e_n)$$

where the starred expressions will possibly (if they are not already proper maplets) be substituted by their purified versions in predicate P. Purification is done by the purify operator:

$$\text{purify}\,(e) = \begin{cases} \text{purify}\,(e_0) \mapsto \text{purify}\,(e_1) & \text{if } e = e_0 \mapsto e_1 \\ e & \text{if } e \text{ is a proper maplet} \\ \text{dc}\,(e) & \text{otherwise} \end{cases}$$

Example. The expression

$$(s \mapsto (t \mapsto \{a, b\}) \mapsto 1 + 2$$

becomes after purification

$$(s \mapsto (t \mapsto x_0) \mapsto x_1 \ .$$

Every sub expression that is substituted has to be bound to its substitute. The conjunct of these equality predicates is denoted as:

$$\text{bindings}\,(\text{purify}\,(i))$$

Example. The *bindings* for the previous example are:

$$x_0 = \{a, b\} \wedge x_1 = 1 + 2$$

The conditional quantification operators $\forall_c$ and $\exists_c$ can now be defined as follows, where ξ represents purify (e):

$$\forall_c\ P\,(e^*) = \begin{cases} P & \text{if } e \text{ is a proper maplet} \\ \forall \text{free}\,(\xi) \cdot\ \text{bindings}\,(\xi) \Rightarrow P\,(\xi) & \text{otherwise} \end{cases}$$

$$\exists_c\ P\,(e^*) = \begin{cases} P & \text{if } e \text{ is a proper maplet} \\ \exists \text{free}\,(\xi) \cdot\ \text{bindings}\,(\xi) \wedge P\,(\xi) & \text{otherwise} \end{cases}$$

Example 1: The predicate

$$\exists_c\ \ (a \mapsto (1 \mapsto 2))^* \in S$$

has the meaning

$$\exists x_0, x_1 \cdot\ x_0 = 1 \wedge x_1 = 2 \wedge a \mapsto (x_0 \mapsto x_1) \in S$$

Example 2: The predicate

$$\exists_c\ \ (a \mapsto b)^* \in S$$

doesn't introduce a quantification:

$$a \mapsto b \in S$$

3.2.1 Equality Rules

Since equality is a symmetric relation, all the following rules can also be applied with exchanged operands. The rules are prioritized in the order they are presented.

Decomposed quantification. Some rules introduce new bound identifiers. This needs to be done in a decomposed form. For keeping the rules simple, the following notation is introduced:

$$\forall/\exists X_T \cdot\ P\,(X)$$

stands for:

$$\forall/\exists\ \text{free}\,(\text{dc}\,(e))\cdot\ P\,(\text{dc}\,(e))$$

where $type(e) = T$

Name	Mask	Right	Remark
ER1	$e = e$	$\top$	
ER2	$x^2 \mapsto y^2 = a^2 \mapsto b^2$	$x = a \land y = b$	
ER3	$\text{bool}\,(P) = \text{bool}\,(Q)$	$P \Leftrightarrow Q$	
ER4	$\text{bool}\,(P) = \text{TRUE}$	P	
ER5	$\text{bool}\,(P) = \text{FALSE}$	$\neg P$	
ER6	$x = \text{FALSE}$	$\neg\,(x = \text{TRUE})$	
ER7	$x = \text{bool}\,(P)$	$x = \text{TRUE} \Leftrightarrow P$	
ER8	$x = f(y)$	$y \mapsto x \in f$	
ER9	$s = t$	$\forall X_T \cdot\ X \in s \Leftrightarrow X \in t$	$s = t$ not in goal $\text{type}\,(s) = \mathbb{P}\,(T)$
ER10	$n^3 = \text{card}\,\left(s^2\right)$	$0 \leq n \land (\exists f \cdot\ f \in s' \rightarrowtail\!\!\!\!\!\twoheadrightarrow 1..n')$	n is an identifier
ER11	$n^2 = \min\left(s^2\right)$	$n \in s \land n \leq \min\,(s)$	
ER12	$n^2 = \max\left(s^2\right)$	$n \in s \land \max\,(s) \leq n$	
ER13	$e_l\,(\text{bop}\,(s)) = e_r$	$\forall_c\ e_1\left(\text{bop}\,(s)^*\right) = e_r$	bop is $\begin{cases} min, \\ max, \\ card, \\ bool, \\ f\,() \end{cases}$

3.2.2 Inequality Rules

For these rules, the left and right-hand side of the mask may be exchanged when the operator is reversed.

Name	Mask	Right	Remark
CR1	$a \prec \min(s)$	$\forall x \cdot\ x \in s' \Rightarrow a' \prec x$	
CR2	$\max(s) \prec a$	$\forall x \cdot\ x \in s' \Rightarrow x \prec a'$	
CR3	$\min(s) \prec a$	$\exists x \cdot\ x \in s' \land x \prec a'$	
CR4	$a \prec \max(s)$	$\exists x \cdot\ x \in s' \land a' \prec x$	
CR5	$a \succ b$	$b \prec a$	
CR6	$a(\text{bop}(s)) \prec b$	$\forall_c\ a\left(\text{bop}(s)^*\right) \prec b$	
CR7	$a \prec b(\text{bop}(s))$	$\forall_c\ a \prec b\left(\text{bop}(s)^*\right)$	

where $\prec$ / $\succ$ stands for $<$ or $\leq$ / $>$ or $\geq$, and *bop* for either min, max, card or $f()$.

3.3 Belongs Predicates

Figure 2 on the next page shows the situation after having applied the basic and comparison rules.

Rule IR3 Purifies the left-hand side of a belongs predicate whose right-hand side is an identifier.

Rule IR3' Purifies the left-hand side of a belongs predicate when it contains a functional image of a Cartesian product type (i.e., hides a maplet expression).

IR1 - IR3' Have to be applied in the order below and before all the other membership rules.

Name	Mask	Right	Remark
IR1	$e \in s$	$\top$	if s is a type expression
IR2	$e \in \mathbb{P}(t)$	$\forall X_T \cdot\ X \in e' \Rightarrow X \in t'$	$\mathbb{P}(T) = \text{type}(e)$
IR2'	$e \in s \leftrightarrow t$	$\forall X_T \cdot\ X \in e' \Rightarrow X \in s' \times t'$	$\mathbb{P}(T) = \text{type}(e)$
IR3	$e^2 \in f$	$\exists_c\ e^* \in f$	f is an identifier
IR3'	$e \in f$	$\forall_c\ e^* \in f$	e is not fully decomposed into maplets

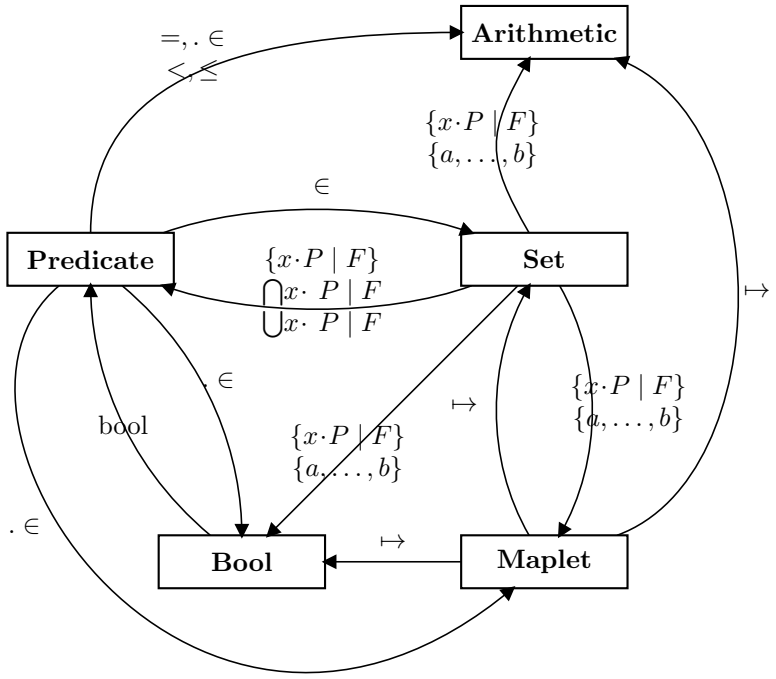

Figure 2: Borders of the formula structure after having basic and comparison rules applied.

_func The following abbreviation is introduced to specify, that a given relation f is a function:

$$\_\text{func}\left(f^2\right) := \forall A_{T_1}, B_{T_2}, C_{T_2} \cdot\ A \mapsto B \in f \land A \mapsto C \in f \Rightarrow B = C$$

where

$$\text{type}\left(f\right) = \mathbb{P}\left(T_1 \times T_2\right)$$

3.3.1 Rules for Left-hand Sides of Arbitrary Form

Name	Mask	Right	Remark
IR4	$e \in \mathbb{N}$	$0 \leq e$	
IR5	$e \in \mathbb{N}_1$	$0 < e$	
IR6	$e^2 \in \left\{x \cdot P \mid f^2\right\}$	$\exists x \cdot\ P\ \land\ e' = f$	
IR7	$e \in (\bigcap x \cdot\ P \mid f)$	$\forall x \cdot\ P \Rightarrow e' \in f$	
IR8	$e \in (\bigcup x \cdot\ P \mid f)$	$\exists x \cdot\ P\ \land\ e' \in f$	
IR9	$e \in \text{union}\left(s\right)$	$\exists x \cdot\ x \in s'\ \land\ e' \in x$	

Name	Mask	Right	Remark
IR10	$e \in \mathrm{inter}\,(s)$	$\forall x \cdot\ x \in s' \Rightarrow e' \in x$	
IR11	$e \in \varnothing$ $e \in \{\}$	$\bot$	
IR12	$e \in r\,[w]$	$\exists X_T \cdot\ X \in w'\ \wedge\ X \mapsto e' \in r'$	$\mathbb{P}\,(T) = \mathrm{type}\,(\mathrm{dom}\,(r))$
IR13	$e \in f\,(w)$	$\exists X_T \cdot\ w' \mapsto X \in f'\ \wedge\ e' \in X$	$T = \mathbb{P}\,(\mathrm{type}\,(e))$
IR14	$e \in \mathrm{ran}\,(r)$	$\exists X_T \cdot\ X \mapsto e' \in r'$	$\mathbb{P}\,(T) = \mathrm{type}\,(\mathrm{dom}\,(r))$
IR15	$e \in \mathrm{dom}\,(r)$	$\exists X_T \cdot\ e' \mapsto X \in r'$	$\mathbb{P}\,(T) = \mathrm{type}\,(\mathrm{ran}\,(r))$
IR16	$e^n \in \{a_1, \ldots, a_n\}$	$e = a_1\ \vee\ \ldots\ \vee\ e = a_n$	
IR17	$e^2 \in \mathbb{P}_1\,(s)$	$e \in \mathbb{P}\,(s)\ \wedge\ (\exists X_T \cdot\ X \in e')$	$\mathbb{P}\,(T) = \mathrm{type}\,(e)$
IR18	$e^2 \in a..b$	$a \leq e\ \wedge\ e \leq b$	
IR19	$e^2 \in s \setminus t$	$e \in s\ \wedge\ \neg\,(e \in t)$	
IR20	$e^n \in s_1 \cap\ \ldots\ \cap s_n$	$e \in s_1\ \wedge\ \ldots\ \wedge\ e \in s_n$	
IR21	$e^n \in s_1 \cup\ \ldots\ \cup s_n$	$e \in s_1\ \vee\ \ldots\ \vee\ e \in s_n$	
IR23	$e^3 \in s^2 \nleftrightarrow t$	$e \in s \leftrightarrow t\ \wedge\ s \subseteq \mathrm{dom}\,(e)$	
IR24	$e^2 \in s \leftrightarrow\!\!\!\!\rightarrow t^2$	$e \in s \leftrightarrow t\ \wedge\ t \subseteq \mathrm{ran}\,(e)$	
IR25	$e^4 \in s^2 \nleftrightarrow\!\!\!\!\rightarrow t^2$	$e \in s \nleftrightarrow t\ \wedge\ t \subseteq \mathrm{ran}\,(e)$	
IR26	$e^8 \in s^2 \rightarrowtail\!\!\!\!\twoheadrightarrow t^2$	$e \in s \twoheadrightarrow t\ \wedge\ \_\mathrm{func}\left(e^{-1}\right)$	
IR27	$e^6 \in s^2 \rightarrow\!\!\!\!\twoheadrightarrow t^2$	$e \in s \rightarrow t\ \wedge\ t \subseteq \mathrm{ran}\,(e)$	
IR28	$e^5 \in s \twoheadrightarrow\!\!\!\!\twoheadrightarrow t^2$	$e \in s \twoheadrightarrow t\ \wedge\ t \subseteq \mathrm{ran}\,(e)$	
IR29	$e^7 \in s^2 \rightarrowtail t$	$e \in s \rightarrow t\ \wedge\ \_\mathrm{func}\left(e^{-1}\right)$	
IR30	$e^6 \in s \twoheadrightarrow\!\!\!\!\rightarrowtail t$	$e \in s \twoheadrightarrow t\ \wedge\ \_\mathrm{func}\left(e^{-1}\right)$	
IR31	$e^5 \in s^2 \rightarrow t$	$e \in s \twoheadrightarrow t\ \wedge\ s \subseteq \mathrm{dom}\,(e)$	

Name	Mask	Right	Remark
IR32	$e^4 \in s \nrightarrow t$	$e \in s \leftrightarrow t \ \wedge \ \_\mathrm{func}\,(e)$	

3.3.2 Rules for Left-hand Side of Form $e \mapsto f$

Name	Mask	Right	Remark
IR33	$e \mapsto f \in s \times t$	$e \in s \ \wedge \ f \in t$	
IR34	$e \mapsto f \in r_1^1 \lhd \ldots \lhd r_n^n$	$\begin{array}{ll} e \mapsto f & \in r_n \vee \\ e \mapsto f & \in \mathrm{dom}\,(r_n) \lhd r_{n-1} \vee \\ e \mapsto f & \in \mathrm{dom}\,(r_n) \cup \mathrm{dom}\,(r_{n-1}) \lhd r_{n-2} \vee \\ & \vdots \\ e \mapsto f & \in \mathrm{dom}\,(r_n) \cup \ldots \cup \mathrm{dom}\,(r_2) \lhd r_1 \end{array}$	
IR35	$e \mapsto f^2 \in r \rhd t$	$e \mapsto f \in r \ \wedge \ \neg\,(f \in t)$	
IR36	$e^2 \mapsto f \in s \lhd r$	$e \mapsto f \in r \ \wedge \ \neg\,(e \in s)$	
IR37	$e \mapsto f^2 \in r \rhd t$	$e \mapsto f \in r \ \wedge \ f \in t$	
IR38	$e^2 \mapsto f \in s \lhd r$	$e \mapsto f \in r \ \wedge \ e \in s$	
IR39	$e \mapsto f \in \mathrm{id}$	$e = f$	
IR40	$e \mapsto f \in r_1; \ldots; r_n$	$\begin{array}{rl} \exists X_{1\,T_1}, \ldots, X_{n-1\,T_{n-1}} \cdot e \mapsto X_1 & \in r_1 \wedge \\ X_1 \mapsto X_2 & \in r_2 \wedge \\ & \vdots \\ X_{n-1} \mapsto f & \in r_n \end{array}$	(1)
IR41	$e \mapsto f \in r_1 \circ \ldots \circ r_n$	$e \mapsto f \in r_n; \ldots; r_1$	
IR42	$e \mapsto f \in r^{-1}$	$f \mapsto e \in r$	
IR47	$e \mapsto f \in \mathrm{pred}$	$e = f + 1$	
IR48	$e \mapsto f \in \mathrm{succ}$	$f = e + 1$	

Remarks:

Nr	Remark
1	$\begin{array}{ll} \mathbb{P}\,(T_1) & = \mathrm{type}\,(\mathrm{ran}\,(r_1)) \\ & \vdots \\ \mathbb{P}\,(T_{n-1}) & = \mathrm{type}\,(\mathrm{ran}\,(r_{n-1})) \end{array}$

3.3.3 Rules for Left-hand Sides of Form $(e \mapsto f) \mapsto g$

Name	Mask	Right	Remark
IR43	$(e \mapsto f) \mapsto g \in \mathrm{prj}_1$	$e = g$	
IR44	$(e \mapsto f) \mapsto g \in \mathrm{prj}_2$	$f = g$	

3.3.4 Rules for Left-hand Sides of Form $e \mapsto (f \mapsto g)$

Name	Mask	Right	Remark
IR45	$e^2 \mapsto (f \mapsto g) \in p \otimes q$	$e \mapsto f \in p \ \wedge \ e \mapsto g \in q$	

3.3.5 Rules for Left-hand Sides of Form $(e \mapsto f) \mapsto (g \mapsto h)$

Name	Mask	Right	Remark
IR46	$(e \mapsto f) \mapsto (g \mapsto h) \in p \parallel q$	$e \mapsto g \in p \ \wedge \ f \mapsto h \in q$	

4 Predicate Simplifications

This section presents some very basic predicate simplification rules, which are applied to the result of the translation process in order to simplify unnecessary complications introduced by the translator.

Name	Mask	Right	Remark
PR1	$\ldots \wedge \bot \wedge \ldots$	$\bot$	
PR2	$\ldots \vee \top \vee \ldots$	$\top$	
PR3	$\ldots \wedge c_{k-1} \wedge \top \wedge c_{k+1} \wedge \ldots$	$\ldots \wedge c_{k-1} \wedge c_{k+1} \wedge \ldots$	
PR4	$\ldots \vee c_{k-1} \vee \bot \vee c_{k+1} \vee \ldots$	$\ldots \vee c_{k-1} \vee c_{k+1} \vee \ldots$	
PR5	$P \Rightarrow \top$	$\top$	
PR6	$\bot \Rightarrow P$	$\top$	
PR7	$\top \Rightarrow P$	P	
PR8	$P \Rightarrow \bot$	$\neg P$	
PR9	$\neg\top$	$\bot$	
PR10	$\neg\bot$	$\top$	
PR11	$\neg\neg P$	P	
PR12	$P \Leftrightarrow P$	$\top$	
PR13	$P \Leftrightarrow \top$	P	(1)
PR14	$P \Leftrightarrow \bot$	$\neg P$	(1)
PR15	$\forall/\exists x_1, \ldots, x_n \cdot \top$	$\top$	
PR16	$\forall/\exists x_1, \ldots, x_n \cdot \bot$	$\bot$	

Remarks:

Nr	Remark
1	applicable with exchanged operands

Name	Mask	Right	Remark
AR1	$\ldots \land P \land \ldots \land P \land \ldots$	$\ldots \land P \land \ldots \land \ldots$	
AR2	$\ldots \lor P \lor \ldots \lor P \lor \ldots$	$\ldots \lor P \lor \ldots \lor \ldots$	
AR3	$\ldots \land P \land \ldots \land \neg P \land \ldots$	$\bot$	
AR4	$\ldots \lor P \lor \ldots \lor \neg P \lor \ldots$	$\top$	
AR5	$P \Rightarrow P$	$\top$	
AR6	$\neg P \Rightarrow P$	P	
AR7	$P \Rightarrow \neg P$	$\neg P$	
AR8	$P \Leftrightarrow \neg P$	$\bot$	(1)
AR9	$\ldots \land Q \land \ldots \Rightarrow Q$	$\top$	
AR10	$\ldots \land Q \land \ldots \Rightarrow \neg Q$	$\neg(\ldots \land Q \land \ldots)$	
AR11	$\ldots \land \neg Q \land \ldots \Rightarrow Q$	$\neg(\ldots \land \neg Q \land \ldots)$	
AR12	$\forall x \cdot P \land \ldots \land Q$	$(\forall x \cdot P) \land \ldots \land (\forall x \cdot Q)$	
AR13	$\exists x \cdot P \lor \ldots \lor Q$	$(\exists x \cdot P) \lor \ldots \lor (\exists x \cdot Q)$	
AR14	$\exists x \cdot P \Rightarrow Q$	$(\forall x \cdot P) \Rightarrow (\exists x \cdot Q)$	

Table 1: Additional predicate simplification rules.

5 Translation Options

In order to reuse this translator for integrating SMT solvers into the Rodin platform, it is necessary to adapt slightly the translation scheme and the goal language. For this purpose, an option mechanism has been added to the translator which allows to tailor the translation to specific needs.

The available options are:

`expandSetEquality` makes rule ER9 applicable to all set equalities, lifting the restriction to sets that are types. Consequently, production pred-setequ is also removed from the goal language.

`aggressiveSimplification` makes the translator simplify more aggressively the predicate obtained after translation. The rules of Table 1 are applied to the result of the translation in addition to the regular PR rules.

6 Architecture

This section describes the overall architecture of the implementation of the translator, which consists of three packages:

- Package `org.eventb.pp` provides the API to the predicate translation. It contains only one abstract class, named `Translator`. This is the only class visible to clients.
- Package `org.eventb.pp.translator` contains the actual implementation of the predicate translator. A small overview is given in the next subsection.
- Package `org.eventb.pp.tests` contains unit tests for the API methods of the translator.

6.1 Translator Implementation

The class hierarchy of the translator is the following:

```
IdentityTranslatorBase
+-- IdentityTranslator
|   +-- BoundIdentifierDecomposition
|   +-- ExpressionExtractor
|   +-- PredicateSimplification
|   \-- Translator
+-- DecomposedQuant
|   \-- Decomp2PhaseQuant
|       +-- ConditionalQuant
|       \-- MapletDecomposer
+-- FormulaConstructor
+-- FreeIdentifierDecomposition
+-- GoalChecker
\-- Reorganizer
```

Next, a small explanation for each class is given.

IdentityTranslatorBase (`IdentityTranslatorBase.java`)
Contains the logic of the IdentityTranslator. For more details, see the description of class `IdentityTranslator` below.

IdentityTranslator (`IdentityTranslator.t`)
Contains the pattern matching part of the identity translation. The idea behind identity translation is the following: The `pp` package contains several recursively implemented translators. These translators do just in some small cases transform a formula part but more often call themselves recursively on the child formulas. These recursive calls are factored into the `IdentityTranslator`.

BoundIdentifierDecomposition (`BoundIdentifierDecomposition.java`)
Implements the decomposition of bound identifiers.

ExpressionExtractor (`Reorganizer.java`)
Extracts min, max, card and $f()$ constructs from arithmetic expressions.

PredicateSimplification (`PredicateSimplification.t`)
Simplifies predicates with rules PR1 - PR16.

Translator (`Translator.t`)
Implements the predicate reduction rules: BR1-BR7, ER1-ER13, CR1-CR7.

DecomposedQuant (`DecomposedQuant.t`)
Implements the decomposed quantification.

Decomp2PhaseQuant (`Decomp2PhaseQuant.java`)
Specializes the decomposed quantification. Adding quantifiers and pushing expressions may now be intermixed. This comfort is paid by doing the whole work twice, once in phase one and again in phase two.

ConditionalQuant (`ConditionalQuant.java`)
Implements the conditional quantification ($\forall_c$, $\exists_c$)

MapletDecomposer (`MapletDecomposer.java`)
Implements decomposition of expressions of Cartesian product type.

FormulaConstructor (`FormulaConstructor.java`)
Implements some helper methods to construct and simplify predicates.

FreeIdentifierDecomposition (`FreeIdentifierDecomposition.java`)
Implements the decomposition of free identifiers.

GoalChecker (`GoalChecker.t`)
Checks whether a given predicate is reduced.

Reorganizer (`Reorganizer.java`)
Is a facade to the ExpressionExtractor.