A Joint Development of Coloured Petri Nets and the B Method in Critical Systems

Pengfei Sun, Philippe Bon, Simon Collart-Dutilleul

Journal of Universal Computer Science, vol. 21, no. 12 (2015), 1654-1683
submitted: 16/2/15, accepted: 1/10/15, appeared: 1/12/15 © J.UCS

A Joint Development of Coloured Petri Nets and the B Method in Critical Systems

Pengfei Sun
(Univ Lille Nord de France, F-59000 Lille, France
IFSTTAR, COSYS/ESTAS
pengfei.sun@ifsttar.fr)

Philippe Bon
(Univ Lille Nord de France, F-59000 Lille, France
IFSTTAR, COSYS/ESTAS
philippe.bon@ifsttar.fr)

Simon Collart-Dutilleul
(Univ Lille Nord de France, F-59000 Lille, France
IFSTTAR, COSYS/ESTAS
simon.collart-dutilleul@ifsttar.fr)

Abstract: Model transformation is an interesting task, which could take advantage of several modelling languages, and meanwhile should respect all the safety requirements. The presented work studies the translation from a valid design solution to a valid implementation, which is a mapping method from coloured Petri nets to abstract *B* machines. Both modelling languages are well known formal methods in the context of safety requirement engineering. The Petri nets are widely accepted by French railway engineers because of a fine graphic representation and their dynamic analysis properties. The *B* machine offers verified software development based on *B* language, which has already been applied in some safety-critical systems. The proposed model translation technique will help to bridge the gap between these two formal methods. This paper shows the systematic process of the translation, which is also illustrated by several case studies. The limitations and future efforts are discussed at the end of the paper.
Key Words: Coloured Petri nets *B* method, modelling languages translation, critical system
Category: J.6, J.7

1 Introduction

The aim of this paper is to describe a process of designing and assessment. On the one hand a fine behavioural specification has to be able to be assessed by experts. On the other hand, the implementation result should respect some common industrial constraints. To save efforts and reduce errors, this paper presents a transformation method from the coloured Petri nets to abstract *B* machines. This method could assist people in quickly shifting from a valid design

solution to a valid input of B development process in the design phase. In this paper we mainly focus on the transformation process and do not deal with the refinement of B machine.

In the French railway field, the Petri nets and the B machine are two industry recognized tools. For instance, the French national railway company (SNCF) is interested in Petri nets and their high-level extensions. They are used as formal tools both in scientific research and practice applications: in the high-level systems, coloured Petri nets are used to assess the performance of European railway signalling systems and the French signalling system [Lalouette et al. 2010, Buchheit et al. 2011]. In the low level systems, Petri nets have been used to develop a new-type railway interlocking control system by M. Antoni [Antoni 2009; 2012], an expert engineer of SNCF. The reason that experienced engineers prefer to use the Petri nets and other high-level extensions is the power of their languages, which have the ability to express a complex system with a compact size. In addition, as graphic languages, the Petri nets have the same advantage as UML, that is having an easy understanding formalism for communicating. Therefore, many practical systems and valid solutions are specified with Petri nets and other high-level Petri nets.

Nevertheless, for urban railway systems, which are more independent and closed systems, the B method has been accepted and has been applied in some key components. Some early success stories are: SAET-METEOR [Behm et al. 1999], a driverless train automation and operation system in metro Line 14 in Paris in 1998; Roissy VAL, a section automatic pilot system for light driverless shuttle for Paris-Roissy airport in 2006, and now is operating in Taipei, Toulouse, Rennes and Turin [Erbin and Soulas 2003]. A recent application is the COPPILOT system [Patin et al. 2006, Lecomte 2008], which is a metro platform screen door controlling system of the ClearSy company. It has been installed in the Paris metro and the Sao Paulo metro. The safety and robustness of B language developed systems convince people of the reliability of the B method, because the final implementation code generated from abstract B machine is considered, and proved to be, safe. So, in the French railway context, B proved model is accepted as a strong safety proof [Boulanger 2013a;b].

Actually, besides the above-mentioned advantages, both languages have some deficiencies in the industrial practices. For large scale, complex systems, the high-level Petri nets could provide a good formal framework and present a concise model [Sun et al. 2014]. So, there are already many valid solutions modelled by Petri nets. However, the assessment processes will probably encounter the "State explosion" problem. Furthermore, there is a big gap between Petri net models and the final implementations, because there are rarely any commercial tools that can formally assess this process. In contrast, the B method is well suited for formal assessment in the development of implementable codes. But its

language demands a lot of pre-training for set theory and first order logic, which greatly increases the research and development costs. Besides, its notations and its commercial tools are not user friendly. So, it has only been implemented in the vital safety-critical applications.

This technique mismatch leads to the following question. Let us assume in the design phase, a model has been validated by industry experts, using dedicated industrial tools such as Petri nets.The problem is how to prove it formally with respect to the specifications and automatically generate the implementable codes. To be more precise, the problem to be solved is the translation from the coloured Petri net formalism into the abstract B machine formalism. The motivations of such a transformation are the followings: from the engineering point of view, two formal methods have some advantages and disadvantages, and we consider them as complementary in the French railway context. This proposition leads us to take advantage of both methods using the techniques of model driven engineering. So, an interesting solution should be a model transformation: from CPN models to the abstract B machines. Such a transformation can build a bridge between critical tasks, from a strong requirement analysis towards a valid implementation on a real system. Last but not least, if we want to validate safety at a system level, this transformation is a strong contribution in order to integrate information from different parts.

Some similar works of this approach contribute efforts from different perspectives. The work in [Korečko et al. 2007, Korečko and Sobota 2014] presents a mapping method of low level Petri nets (the Petri nets with undistinguishable tokens) to abstract B machines and to Event B machines. A similar work [Attiogbe 2009] presents an encoding of PT nets to the *Event-B* language. The work of [Bon 2000, Bon and Dutilleul 2013] successfully translates a non-hierarchical CP-net with numerical colour sets to B machines with Atelier B syntax. Nevertheless, the practical experience shows this work generates a lot of unproven POs using automatic prover in Atelier B. In this paper, our research is an improvement of the work from [Bon and Dutilleul 2013]. We continue to use their basic data structure definitions. Based on this, we introduce a more explicit framework of the translation, simplify the operations for multi-set structures, make the translation compatible with non-numeric types of colour sets. Then, we reform the mechanism of multi-set specification, in order to ensure the target B machines can be automatically proved by Atelier B without using the interactive prover.

The rest of the paper is organized as follows: sections 2 and 3 provide some necessary formal definitions about coloured Petri nets and B language. Note that the main contribution of these two sections is to present the transformation from coloured Petri nets to B machines. Consequently, the presentations of the two formal methods are efficient rather than exhaustive. Section 4 presents the mapping methodology from non-hierarchical coloured Petri nets to abstract B

machines. In the conclusion, we discuss the limitations of the current method and propose some further ideas for research.

2 Coloured Petri Nets

Petri nets are a mathematical modelling language and were first developed by C.A Petri in order to describe distributed systems. Petri nets provide the foundation of a straightforward graphical notion for specifications of stepwise processes, such as selections, loops, and parallel operations. Moreover, all the executions of Petri nets are performed by a rigorous and explicit algorithm process, which is based on a complete mathematical theory. So, they could be used for some formal proofs.

Coloured Petri nets (CP-nets) are high-level Petri nets, developed by Prof. Kurt Jensen [Jensen 1987]. They are a backward compatible extension of Petri nets which have the feature of performing a high-level programming language. The notation of CP-net is called *CPN Meta language*, whose language foundation follows the Standard ML [Milner 1997]. CPN Meta language provides basic data types, data constructors and pre-defined functions in order to develop a compact model with strong performance capabilities. CP-net maintains the properties of classical Petri nets and provides a new notation of *token* to mark the difference between them. Then, each token is no longer noted with indistinguishable black dots, but with a data value which is named as the *token colour*. Each place in CP-net models can be marked with different tokens, but with the same type which is specified from programming languages. The type of each place is called the *colour set*. The state of a CP-net model (*marking*) is composed of all the tokens, including both their numbers and their colours. Although the CP-nets accept some programming language, their syntax and semantics are still based on rigorous mathematical definitions. So, CP-nets are considered as a formal method.

The following subsection gives the necessary definitions of coloured Petri nets. They are quoted from the book of Kurt Jensen (2009) "Coloured Petri nets: modelling and validation of concurrent systems", if the readers already have a background in coloured Petri net language, they may skip these subsections.

2.1 Non-hierarchical coloured Petri net [Jensen and Kristensen 2009]

The formal definition of a non-hierarchical and untimed CP-net is given as below.

Definition 1. A non-hierarchical coloured Petri net is a nine-tuple $CPN = (P, T, A, \Sigma, V, C, G, E, I)$, where:

(i) P is a finite set of **places**.

(ii) T is a finite set of **transitions**, $P \cap T = \phi$.

(iii) A is a finite set of **arcs**, $A \subseteq P \times T \cup T \times P$.

(iv) Σ is a finite set of **colour sets**, $\Sigma \neq \phi$.

(v) V is a finite set **typed variables**, $Type[v] \in \Sigma$ for all variables $v \in V$.

(vi) C is the **colour set function**, $C : P \to \Sigma$

(vii) G is the **guard function**, $G : T \to EXPR_v$ and for $\forall t \in T$ that $Type[G(t)] = Bool \wedge Type[Var(G(t))] \subseteq \Sigma$.

(viii) E is the **arc expression function**, $E : A \to EXPR_v$ and for $\forall a \in A$ that $Type[E(a)] = C(p)_{\mathrm{MS}} \wedge Type[Var(E(t))] \subseteq \Sigma$.

(ix) I is the **initialisation function**, $I : P \to EXPR_\phi$ and for $\forall p \in P$ that $Type[I(p)] = C(p)_{\mathrm{MS}}$.

To formally define the behaviour of CP-nets, some semantic concepts and notations should be introduced.

Definition 2. For a CP-net, the following concepts exists:

(i) A **marking** M is a mapping from places into multi-set of tokens, for $\forall p \in P : M(p) \in C(p)_{\mathrm{MS}}$,

(ii) The **initial marking** M_0 is defined as $\forall p \in P : M_0(p) = I(p)\langle\rangle$.

(iii) The **variables of a transition** t is denoted as $Var(t) \subseteq V$ and $Var(t) = \{v \mid v \in Var(G(t)) \vee \exists a \in A : v \in Var(E(a))\}$.

(iv) A **binding** of a transition t is a function b that maps each variable $v \in Var(t)$ into a value $b(v) \in Type[v]$. The set of all bindings for t is denoted by $B(t)$.

(v) A **binding element** is a couple (t, b) where $t \in T$ and $b \in B(t)$. The set of all binding elements in a CP-net is denoted as BE.

(vi) An **arc expression** is $E(p, t)$ or $E(t, p)$ denote the arc expression on the input or output arc from p to t. In an enabled binding element (t, b), the multi-set of tokens removed from an input place p when t occurs in a binding b is given by $E(p, t)\langle b\rangle$, and $E(t, p)\langle b\rangle$ is the multi-set of tokens added to an output place p.

The behaviours of a CP-net are the transfer of tokens, which are based on the enabling rules of the binding elements, there are represented as following.

Definition 3. In the marking M, a binding element $(t, b) \in BE$ is called **enabled** if and only if the following two conditions are satisfied:

(i) $G(t)\langle b\rangle = ture$

(ii) $\forall p \in P : E(p,t)\langle b\rangle \leq M(p)$.

After firing the enabled transition t, the new marking of the system is:

(iii) $\forall p \in P : M'(p) = (M(p) - -E(p,t)\langle b\rangle) + +E(t,p)\langle b\rangle$.

The operator "++" takes two multi-sets as arguments and returns the union (the sum). In the same way, the operator "−−" the subtraction of two multi-sets.

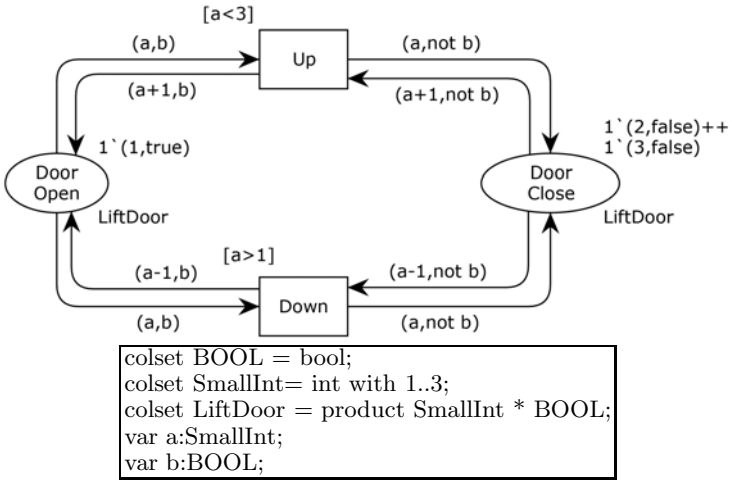


Figure 1: CP-net of an Elevator

To better illustrate the definitions, a small example will be introduced. This is a toy model of an elevator door control system. There is a lift car that can move upwards or downwards between three floors. Tokens in "Door Open" and "Door Close" places represent lift doors of different floors. The token in the form of (a,b) indicates the lift door on the ath floor is open/closed (b=true/b=false). For

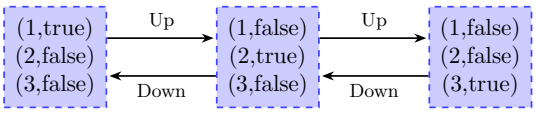


Figure 2: System state of Fig. 1

example, the token (1,true) means the lift door on first floor is open. Its system states can be found in Fig. 2, where each state is represented by the status of all the tokens. The arcs between different states are the fired transitions.

3 *B*-Method

The *B* machine is a model-oriented method, developed by Prof. Jean-Raymond Abrial [Abrial 1996]. It is a well-defined software development method based on *B*, and it has robust, commercially available tool support for the whole development process, such as Atelier B [see 1].

The *B* language provides a high-level specification formalism called abstract *B* machine. The specifications will be manipulated through a progressive development process that evolves the abstract *B* machines into implementable program codes. Each development process is called a *Refinement*, which "refines" the previous specification into a more concrete one. In fact, each refinement is a process that replaces some of the non-deterministic composition with deterministic ones or programming-like forms. A refinement could be further refined until it is deterministic enough and it is called the *Implementation*. The implementation can be automatically transformed into the final code, such as C, ADA or other programming languages. The consistency of all the abstract **B** machines and the correctness of each development step are validated by a set of **proof obligations** (POs). New POs are generated along with the refinement processes in order to enable the B method to build error-free proven systems. In this paper, we mainly focus on the abstract *B* machine, so further information of the refinement and the **B** development can be found in [Abrial 1996].

The notation of the specifications in *B* method is known as the abstract machine notation (AMN). Fig. 3 presents an abstract *B* machine notation. Normally, it could contain different clauses to describe its properties and operations. These clauses can be classified into three categories: composition, declarative and executive [Boulanger 2014]. In Fig. 3, each category is marked with a character style, the compositions are normal style, the declaratives are *italics* style, and the executives are **bold** style.

The compositions are used to define the relationships between different abstract machines which belong to the same system. The possible relations are INCLUDES, SEES, IMPORTS, EXTENDS or USES. Each clause indicates a different visibility and access permission on the components of the other abstract machines. The declarative clauses describe the static state of a given system with a set of data statement (SETS, VARIABLES, CONSTANTS). Then

[1] Atelier B, developed by ClearSy company, is an industrial tool that allows for the operational use of the B Method to develop defect-free proven software (formal software)

```
MACHINE M(p)
SEE N
CONSTRAINTS C
SETS St
CONSTANTS k
PROPERTIES Bh
VARIABLES v
INVARIANT I
DEFINITIONS D
INITIALISATION T
OPERATIONS
  y ← op(x) =
     PRE P THEN S END
  ...
END
```

Figure 3: General structure of B-machine

the consistency of these data is always validated by the clause PROPERTIES for constants and clause INVARIANT for variables. The executive part presents the dynamic actions of an abstract machine which includes the INITIALIZATION and the OPERATIONS. The initialisation is the initial data assignment action. The operations could modify the value of data in order to achieve state changes of a system. The DEFINITIONS clause offers explicit declarations of textual definition of a component, and may be in parameterized form. Their recalls are directly replaced by the corresponding terms during the lexical analysis phase. In Atelier B, definitions can also be included from external files. In an abstract B machine, the static definitions are based on the set theory and first-order predicates, while all the operations are based on the *generalized substitution language* (GSL).

A subset of GSL used in the following sections is shown in Tab. 1, where x denotes a variable, E is an expression, R is a predicate and S,S_1,S_2 are generalized substitutions. Note that the "PRE-THEN-END" structure is only usable if the predicate is valid, whereas the other conditional substitutions are always performed, but their result depends on the validity of a predicate. Detail explanations about GSL can be found in [Abrial 1996, Boulanger 2014].

Note that the abstract B machines are expressed in non-deterministic formalism. Meanwhile, they follow the rules of set language and first order logic. So, some of the algorithms, such as loops and recursion, are forbidden in the abstract B machines. They can only be achieved in refinement and implementation.

Substitution	**Meaning of GS**
x:=E	$[x := E]R \Leftrightarrow$ substitution of all the free occurrences of x in R by E
skip	$[skip]R \Leftrightarrow R$
$S_1 \| \| S_2$	$[S_1 \| \| S_2]R \Leftrightarrow [S_1]R_s \wedge [S_2]R_t$
PRE E THEN S_1 END	If predicate E holds, do S_1, otherwise do anything
SELECT E THEN S_1 END	If E holds, do S_1, otherwise do not execute
IF E THEN S_1 ELSE S_2 END	If E holds, do S_1, otherwise do S_2
VAR v IN S_1 END	For any values of local variables from the list v do S_1
ANY v WHERE E THEN S_1 END	For any values of variables from v that satisfy E do S_1, If no values satisfy E, do not execute

Table 1: A subset of generalized substitutions [Boulanger 2014].

4 Non-hierarchical CP-net to *B* machine Translation

With the extensive applications of formal methods, sometimes the integration of formal languages is unavoidable. The translation from Petri nets to B machine seems to have been a research interest in recent years. Some similar work of this approach can be found in [Korečko et al. 2007, Korečko and Sobota 2014, Attiogbe 2009]. Our research is an improvement of the work from [Bon and Dutilleul 2013]. In this paper, we introduce an explicit and detailed transformation framework, providing an automatically proved B machine. Let us remark that the assessment of high-level design is a major safety element on its own. In this case, you may not need to execute the complete B refinement processes. Consequently, in this paper, we do not really focus on the B refinement.

The translation introduced in this section is for non-hierarchical CP-nets. It will consist of 4 parts. First, the framework (or template) of the translation is presented, for mapping the Petri net's "Place-transition" structure to the B machine formalism. Second, the colour property of CP-nets is achieved by introducing the multi-set mechanism, which defines the token representation and its mathematical operators. It allows binding tokens with different colour types to each places, and modifying the content of the tokens. Then, different colour sets in CP-nets are mapped into abstract machine notation and user-define declarations are discussed. Finally, two non-hierarchical CP-net models are introduced to demonstrate the translation process.

MACHINE M
SETS $Colset_1, Colset_2, \cdots, Colset_{k-1}$
CONSTANTS $Colset_k, Colset_{k+1}, \cdots, Colset_l$
PROPERTIES
$Colset_k := EXPR_V^k$ & $Colset_{k+1} := EXPR_V^{k+1}$ & $\cdots$
& $Colset_l := EXPR_V^l$
VARIABLES $VP_1, VP_2, \cdots, VP_m$
INVARIANT
$VP_1 : (Colset_1)_{\mathrm{MS}}$ & $VP_2 : (Colset_2)_{\mathrm{MS}}$ & $\cdots$ & $VP_m : (Colset_l)_{\mathrm{MS}}$
INITIALISATION
$VP_1 := VI_1 \parallel VP_2 := VI_2 \parallel \cdots \parallel VP_m := VI_m$
DEFINITIONS
"MultiSets.def";
...
VAR_t_j == $EXPR_V^{t_j}$; // For trainsition T_j
E_p_1_t_j == $EXPR_{\mathrm{VAR\_}t_j}$;
E_p_2_t_j == $EXPR_{\mathrm{VAR\_}t_j}$;
...
E_t_j_p_1 == $EXPR_{\mathrm{VAR\_}t_j}$;
E_t_j_p_2 == $EXPR_{\mathrm{VAR\_}t_j}$;
...
GRD_t_j == $EXPR_V$;
CDT_t_j == $EXPR_V$;
...
OPERATIONS
OP_1= SELECT CDT_1 THEN ACT_1 END;
OP_2= SELECT CDT_2 THEN ACT_2 END;
...
OP_n= SELECT CDT_n THEN ACT_n END;
END

Figure 4: The framework for translation of a B machine

4.1 Framework of non-hierarchical translation

For a CP-net $N = (P, T, A, \Sigma, V, C, G, E, I)$, we define that $P = \{p_1, ..., p_m\}$, $T = \{t_1, ..., t_n\}$, $A = \{(p,t) \mid p \in P \wedge t \in T\} \cup \{(t,p) \mid t \in T \wedge p \in P\}$, $\Sigma = \{\sigma_1, ..., \sigma_{k-1}, \sigma_k, ..., \sigma_l\}$, $C = \{c_1, ..., c_m\}$, $G = \{G(t) \mid t \in T\}$ and $I = \{I(p) \mid p \in P\}$. Let β a the mapping $\beta : CPN \to AMN$, then the image of CP-net N under β is a B machine Bm, $Bm = \beta[N]$ with the following structure.

For all the elements mentioned in Fig. 4, they have the following mapping relationship:

(i) "MultiSets.def" is an external definition file, which will be introduced in the next subsection.

For each colour set $\forall k \in \{1..l\}$, the mapping relations are shown below.

(ii) $Colset_k = \beta[\sigma_k]$ is the colour set , where $\sigma_k = C(p_k) \in \Sigma$, C is the colour set function in Definition 1.

(iii) $(Colset_k)_{\mathrm{MS}} = \beta[C(p_k)_{\mathrm{MS}}]$, is the multi-set of colour .

For each place related element $\forall i \in \{1..m\}$, the relations are shown as follows.

(iv) $VP_i = \beta[M(p_i)]$, is the marking of each place, where $p_i \in P$, M is the marking in Definition 2.

(v) $VP_i \in (Colset_k)_{\mathrm{MS}}$, where $Type[VP_i] = (Colset_k)_{\mathrm{MS}}$

(vi) $VI_i = \beta[I(p_i)\langle\rangle]$ is the initial value , where $Type[VI_i] = (Colset_k)_{\mathrm{MS}}$.

(vii) The initial marking M_0 is defined as $M_0 = \sum_{i=1}^{m} VI_i$

The detail definitions are that for $\forall p, t$ $(p \in P, t \in T)$, we have:

(viii) E_a is the arc expression, where $E_{p,t} = \beta[E(p,t)]$ and $E_{t,p} = \beta[E(t,p)]$.

(ix) $E_a = \begin{cases} EXPR_t \in (Colset_k)_{\mathrm{MS}} & if\ a \in A \\ \Phi_{\mathrm{MS}} & if\ a \notin A \end{cases}$

(x) $cdt(p,t) = \begin{cases} VP \geq E_{p,t} & if\ p \in {}^*t \\ \mathsf{true} & if\ p \notin {}^*t \end{cases}$

(xi) $act(p,t) = \begin{cases} VP := VP - -E_{p,t} + +E_{t,p} & if\ p \in {}^*t \cup t^* \\ \mathsf{skip} & if\ p \notin {}^*t \cup t^* \end{cases}$

For each transition related element $\forall j \in \{1..n\}$, the relations are shown below.

(xii) $OP_j = \beta[t_j]$ is the transition, where $t_j \in T$.

(xiii) $VAR_j = \beta[Var(t_j)]$ is the variables of transition t_j.

(xiv) $GRD_j = \beta[G(t_j)]$ is the guard function.

(xv) CDT_j is the condition of each operation, $CDT_j = (VAR_j \in Colset_k) \wedge cdt(p_1, t_j) \wedge cdt(p_2, t_j) \wedge \cdots \wedge cdt(p_m, t_j) \wedge \beta[G(t_j)]$.

(xvi) $ACT_j = act(p_1, t_j) \wedge act(p_2, t_j) \wedge ... \wedge act(p_m, t_j)$.

In general, the clauses SETS, CONSTANTS and PROPERTIES define all the new sets used in the model. Each element of VARIABLES VP in the B machine M represents the marking $M(p)$ of given place p in the CP-net N. The INVARIANT clause assigns a multi-set of different colours to each place. The INITIALISATION clause ensures each place has the same initial value as $m_0(p)$. The DEFINITIONS clause prepares all the variables of transition, arc expression functions and guard functions for the following operations. In OPERATIONS, each OP represents a transition t, the predicate CDT is similar to the CP-net enabling condition, and the ACT is the token computation formula. The substitution *SELECT* in each operation is the conditional bounded choice. Its actions can only be executed when its condition CDT is true.

4.2 Multi-set concept

One of the important features of coloured Petri nets is the different data types. A CP-net marking corresponds to a multi-set, which contains information of token numbers and token colours. Each behaviour of the transitions is associated to the marking modifications. In order to integrate this mechanism into the B machine framework, the multi-set specifications should be introduced.

In CP-net ML language, there is a predefined set of basic types named *simple colour sets*. These simple colour sets can be used to compose structured colour sets using a set of *colour set constructors* [Jensen and Kristensen 2009], while in the abstract machine notation, only simple data types are predefined. The composed ones will be presented in the next subsection. The mechanism to be introduced must be able to cope with all types of multi-sets. However, it is both unwise and unnecessary to pre-define all types of multi-sets in a single abstract machine. In order to be more adaptable and flexible, we introduce a series of parameterized definitions, which could associate the token numbers with their colours, and allow the modification of tokens via operations. These definitions are written in a "MultiSets.def" file, which could be directly included in any abstract B machine, and these definitions can help to complement the clauses in the previous framework. Fig. 5 gives the content of the file.

The defined function `MS(ss)` is to give the formalism of the marking in B machines. The function `Ms_Empty(ss)` generates an empty token with colour type ss for the initialization propose and further token operations. The function `Ms_Subset(ms1,ms2,ss)` makes a comparison for multi-sets $ms1$ and $ms2$ which is the same colour type ss, the function returns true if and only if the number of each element in $ms1$ is equal to or greater than that in $ms2$. The function `Ms_Add(ms1,ms2,ss)` and `Ms_Less(ms1,ms2,ss)` are addition and subtraction algorithms for multi-sets.

From a mathematical point of view, part of the preconditions in `Ms_Add` and `Ms_Less` seems redundant, such as $ms1(ee) : ran(ms1)$, $ms2(ee) : ran(ms2)$,

"MultiSets.def"

```
DEFINITIONS
MS(ss)==(ss<->NATURAL);
Ms_Empty(ss)=={elt|elt:ss*{0}}
Ms_Subset(ms1,ms2,ss)==!elt.(elt:ss=> ms1(elt)>=ms2(elt));
Ms_Add(ms1,ms2,ss)==
       (%ee.(ee:ss & ms1(ee):ran(ms1) & ms2(ee):ran(ms2)
       & ms1(ee)>=0 & ms2(ee)>=0 | ms1(ee) + ms2(ee)));
Ms_Less(ms1,ms2,ss)==
       (%ee.(ee:ss & ms1(ee):ran(ms1) & ms2(ee):ran(ms2)
         & ms1(ee)>=0 & ms2(ee)>=0 | ms1(ee) - ms2(ee)));
```

Figure 5: Specifications of multi-sets and their operations

$ms1(ee) >= 0$, $ms2(ee) >= 0$. But they are some important additional theories, which can help the Atelier-B prover to prove the POs. Using this technique, the multi-set mechanism can be automatically proved. It is applicable for all data types.

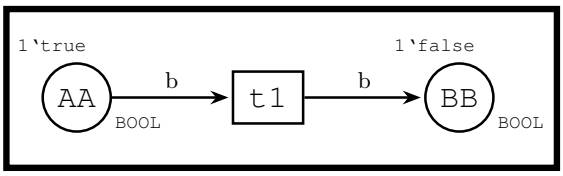


Figure 6: Toy example of Multi-set demonstration

For a better understanding, we take an example of a simple colour set to demonstrate the above-mentioned definitions. In Fig. 6, there is a simple transition with two places. Each place is the same colour type $BOOL$. The initial marking of this CP-net is that place AA has a token $1`true$, while place $BB$ has a $1`false$. After firing the transition $t1$, AA has no token and BB's marking is $1`true++1`false$. In the abstract machine notation, the token representations will be defined as follows.

(i) The variables in the B machine are also denoted with `AA` and `BB`.

(ii) Each variable has a data type, `AA:MS(BOOL)`, `BB:MS(BOOL)`, where `MS(BOOL)` $\Rightarrow \{true|->n, false|->n\}$, $n \in$ NATURAL

(iii) Each variable needs initialization,
`AA:=Ms_Empty(BOOL)<+ true*{1}` $\Rightarrow$ `{true|->1,false|->0}`,

`BB:=Ms_Empty(BOOL)<+ false*{1}` $\Rightarrow$ `{true|->0,false|->1}`.

(iv) After firing transition $t1$,
`AA= Ms_Empty(BOOL)` $\Rightarrow$ `{true|->0,false|->0}`,
`BB= Ms_Empty(BOOL)<+{true, false}*{1}` $\Rightarrow$ `{true|->1, false|->1}`

4.3 Colour sets and CPN declarations

The set of token colours in CP-nets are specified by ML programming language. They are classified into two categories. The basic types which are derived from Standard ML are simple colour sets, and the others are compound colour set using the previously declared colour sets. In this paper only non-time related colours are involved.

In the types supported by the CPN Tools [see 2], the simple colour sets and their corresponding data types in B language are enumerated in the following Tab. 2. Similar data types for Integer, Boolean and Enumerated can be found in abstract machine notation.

Colour set	CPN specification	B notation
Unit	comprises a single element **colset** name = **unit**	no direct corresponding
Boolean	set of true and false **colset** name = **bool**	`BOOL`
Integer	numerals without a decimal point **colset** name = **int**	`INTEGER`
String	sequences of printable characters **colset** name = **string**	`STRING` only in operation input
Enumerated	Enumeration values **colset** name = **with** id0|id1| $\cdots$ |idn	`SET=`$\{E_1, \cdots, E_n\}$
Index	sequences of values comprised of an identifier **colset** name=**index** id **with** exp1..exp2	no direct corresponding

Table 2: Comparison of basic data types

For the other simple colour types, we can only have a non-direct translation. The **Unit** colour sets can be represented with a set containing only one element, such as `unit={new_unit}`. The **Index** colour set is a concrete instance of a positive integer. So, it could be denoted by a set of finite positive integers

[2] CPN Tools is a tool for editing, simulating, and analysing Coloured Petri nets. It is originally developed by Kurt Jensen and his CPN Group at Aarhus University.

`index={int-exp1..int-exp2}`, where `index:NAT`. The **String** colour sets is the set of all text strings, while in the abstract machine notation, the concrete type of "string" can only be used in the operation input parameters. In practical translation, we recommend using a finite enumerate set to indicate all the necessary strings.

For the compound colour sets, CP-nets use constructors to define more complex colour sets.

The **Product** color sets ***colset*** *Name* = ***product*** *name1* $*$ *name2* $*\cdots*$ *namen, where n>=2*. In *B* notation, there is the same Cartesian product '$*$', and definition will be `Name:=name1 * name2 *⋯* namen`.

The **record** colour set ***colset*** *Name* = ***record*** *id1 : name1* $*$ *id2 : name2* $*\cdots*$ *idn : namen*. It is a fixed-length colour set and has the same function as Product. The only difference is that a product colour is position-dependent, while in record colour, each component has a unique label to be identified. In *B* language, the record concept is the same, and the notations are: The set of records is `SET = struct(id1:name1, ⋯, idn:namen)`, an extensive record is `REC = rec(id1:name1, ⋯, idn:namen)`, where n is an integer greater than or equal to 1. The access to a record field (quote operator) is `idi'REC`.

The **List** colour set is a variable-length data type. The values are a sequence whose colour must be the same type, ***colset*** *name* = ***list*** *name0*. As one of the most flexible structures in coloured Petri nets, the list structure has many pre-defined operations which can achieve functions such as inquiry, self-loop and recursion. But this concrete programming language is only accepted in the Implement phase of *B* method. So the mapping of the lists form CP-net to *B* machine is conditional, and the most suitable candidate is "Sequence expressions". Tab. 3 is a partial mapping of two data structures. The other list functions could only be realized inside the operations.

Other pre-defined *list* functions are more programming like functions. They cannot be written in a compact form with set theory and first order logic. In practice, depending on the context, *list* functions may appear in different forms in the operations. Sometimes the function can only be realized in the implementation phase, where recursion and loop are allowed in *B* machines. Here is an example of achieving the same function inside the operation.

CPN specification	***B* notation**

ins_new `l x`
if x is not a number of list l, then x is inserted at the end of l, otherwise l is returned
	`ann<--ins_new(ll,xx)==`
	`IF xx : ran(ll)`
	`THEN aa := ll`
	`ELSE aa := ll<-xx`
	`END`

colset Lst = **list** CS	`Lst:=seq(CS)`
nil or `[]` empty list	`[]`
hd Lst head, the first element of the list	`first(Lst)`
tl Lst tail, the last element of the list	`tail(Lst)`
length Lst length of list	`size(Lst)`
elt**::**List prepend element as head of list	`elt->Lst`
rev Lst reverse list	`rev(Lst)`
List.nth(Lst, nth) nth element in list	`Lst(nth)`
List.take(Lst, n) returns first n elements of list	`Lst/|\n`
List.drop(Lst, n) returns what is left after dropping first n elements of list	`Lst|/n`
List.null Lst returns true if list is empty	`(Lst=[])`
mem Lst elt returns true if element is in the list	`elt : ran(Lst)`
contains Lst1 Lst2 returns true if all elements in list 2 are elements in list 1 (ignoring the multiplicity of elements in list 2)	`ran(Lst1)<:ran(Lst2)`
union Lst1 Lst2 (or Lst1^^Lst2) the concatenation of two lists	`Lst1^Lst2`
ins Lst elt inserts element at the end of list	`Lst<-elt`

Table 3: Mapping of *List* data type

Besides the declaration of colour sets, the CPN Tools also allow constant declaration and user-define function declaration. A constant declaration binds a value to an identifier, which then works as a constant. It should be defined in the *B* constant part. The functions of a *B* machine are directly used in the operations. They can be accomplished by the substitution which is similar to programming language, such as IF condition and Case condition. Some simple parameterized function may appear in the definition as reusable modules.

4.4 Translation Equivalence

Earlier in this section, the methodology of translation from non-hierarchical CP-nets to B machines was defined formally, as well as the multi-set concept and some common token colours. With the translation rules presented, we now present a global design for proving that the model translation is equivalent, also known as strongly consistent. To better illustrate the verification properties, we use the example of Fig. 1 as a study case in this subsection.

4.4.1 Conformance

The basic need is to ensure that the transformed specifications are well-formed with respect to its transformation language. In the CP-nets, the state (or the static property) of the system is represented by the marking, which is the mapping from places into multi-set of tokens. In the transformed specifications, the variables are used to express the system state, which is the marking, $VP == \beta[M(p)]$. The variables obey the same invariants as the marking in CP-nets: the colour set invariant and the initialization invariant.

Definition 4. Let a B machine Bm be the image of a CP-net N after the translation β, where $P = \{p_1, \cdots, p_m\}$. The state of the B machine $State(Bm)$ is defined as follows:

(i) $State(Bm) = VP_1 \wedge \cdots \wedge VP_m$

(ii) $State_0(Bm) = VI_1 \wedge \cdots \wedge VI_m$

Lemma 1. The relation of the system state before and after the translation is shown as:

$$\begin{aligned} State(Bm) &= VP_1 \wedge \cdots \wedge VP_m = \beta[M(p_i)] \wedge \cdots \wedge \beta[M(p_m)] \\ &= \beta[M] \Leftrightarrow M \\ State_0(Bm) &= VI_1 \wedge \cdots \wedge VI_m = \beta[I(p_i)] \wedge \cdots \wedge \beta[I(p_m)] \\ &= \beta[I] \Leftrightarrow I == M_0 \end{aligned}$$

In the example of Fig. 1 there are such transformed specifications:

```
VARIABLES
  OpenDoor, CloseDoor
INVARIANT
  OpenDoor : MS(LiftDoor) &
  CloseDoor : MS(LiftDoor)
INITIALISATION
  OpenDoor := Ms_Empty(LiftDoor) <+ {(1|->TRUE)|->1} ||
  CloseDoor := Ms_Empty(LiftDoor) <+ {(2|->FALSE)
      |->1,(3|->FALSE)|->1}
```

The colour set is related to different data types, and each new type (or non-common type) is pre-defined in the declaration part of CP-nets. Such types should also be declared in the B machines. They will be used as an important part of invariants, because all the variables and operation must comply with their own "colour (data types)" .

```
CONSTANTS
  SmallInt, LiftDoor
PROPERTIES
  SmallInt<<:INT & SmallInt=1..3 &
  LiftDoor=(SmallInt*BOOL)
```

In this way, all the system states have the corresponding images and their possible invariants are held at all time.

4.4.2 Execution Semantics

The dynamic properties of the CP-nets are expressed by firing the transitions, while such properties can only be approached by operations in B machines. According to Definition 3, a transition is called enabled at marking M, if and only if there are some qualified binding elements.

In order to present the enabling conditions, it is necessary to have the variables of the transition. In Definition 2, the variable is defined as $t \in T : Var(t) = \{v \mid v \in Var(G(t)) \vee \exists a \in A : v \in Var(E(a))\}$ In the B machine, variables of a transition are denoted as $VAR_j = \beta[Var(t_j)]$. According to B language syntax, each variable should have its data typing in the condition CDT_j of the operation t_j. We denote the image of a binding $b \in B(t_j)$ as b^β. Then the enabled conditions for the B machines are:

(i) $\beta[G(t)\langle b\rangle = true] \;\Rightarrow\; \beta[G(t)\langle b\rangle] = true$
$\Rightarrow\; \beta[G(t)]\beta[\langle b\rangle] = true \;\Rightarrow\; GRD_j\langle b^\beta\rangle = true$

(ii) $\forall p \in P : \beta[E(p,t)\langle b\rangle \leq M(p)] \;\Rightarrow\; \beta[E(p,t)\langle b\rangle] \leq \beta[M(p)]$
$\Rightarrow\; E_{p,t}\langle b^\beta\rangle \leq VP \;\Rightarrow\; \text{Ms\_Subset}(VP_p, E_{p,t}, Colset_p)\langle b^\beta\rangle.$

So we can claim that the enabling rules of the translation are consistent.

Lemma 2. Let a transition t be enabled by a binding element $(t, b) \in BE$. The image of such transition is $OP_t = \beta[t]$, which could also be enabled by the correspondence condition (t, b^β), as they satisfy the following conditions:

(i) $\beta[G(t)\langle b\rangle = true] \;\Rightarrow\; GRD_j\langle b^\beta\rangle = true$

(ii) $\forall p \in P : \beta[E(p,t)\langle b\rangle \leq M(p)] \;\Rightarrow\; \text{Ms\_Subset}(VP_p, E_{p,t}, Colset_p)\langle b^\beta\rangle.$

Here we have a demonstration of the operation OP_UP, which is the image of transition "UP" in Fig. 1.

```
DEFINITIONS
  "MultiSets.def";

  VAR_UP == aa;
  E_OpenDoor_UP== (Ms_Empty(LiftDoor) <+ {(aa|->TRUE)
      |->1});
  E_UP_OpenDoor== (Ms_Empty(LiftDoor) <+ {((aa+1)|->TRUE)
      |->1});
  E_CloseDoor_UP== (Ms_Empty(LiftDoor)<+ {((aa+1)|->FALSE
      )|->1});
  E_UP_CloseDoor== (Ms_Empty(LiftDoor)<+ {(aa|->FALSE)
      |->1});
  GRD_UP== (aa<3);
  CDT_UP== (aa:SmallInt & GRD_UP &
      Ms_Subset(OpenDoor, E_OpenDoor_UP, LiftDoor) &
      Ms_Subset(CloseDoor,E_CloseDoor_UP,LiftDoor));
  ...
```

If a binding element $(t, b) \in BE$ is enabled in M, it may occur, and it leads to a new marking M'. We say that the marking M' is directly reachable from M, which is also written as $M \xrightarrow{(t,b)} M'$. The image of M' is:

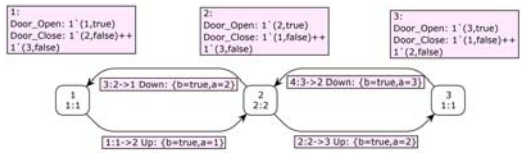
$$\begin{aligned}\forall p \in P : \beta[M'(p)] &= \beta[M(p) - -E(p,t)\langle b\rangle + +E(t,p)\langle b\rangle] \\ &= VP_p - -E_{p,t}\langle b^\beta\rangle + +E_{t,p}\langle b^\beta\rangle \\ &= ACT_t\langle b^\beta\rangle \\ &= \text{Ms\_Add}(\text{Ms\_Subset}(VP_p, E_{p,t}, Colset_p), E_{t,p}, Colset_p)\langle b^\beta\rangle\end{aligned}$$

So, we can also claim the consistence of the occurrence of the translation.

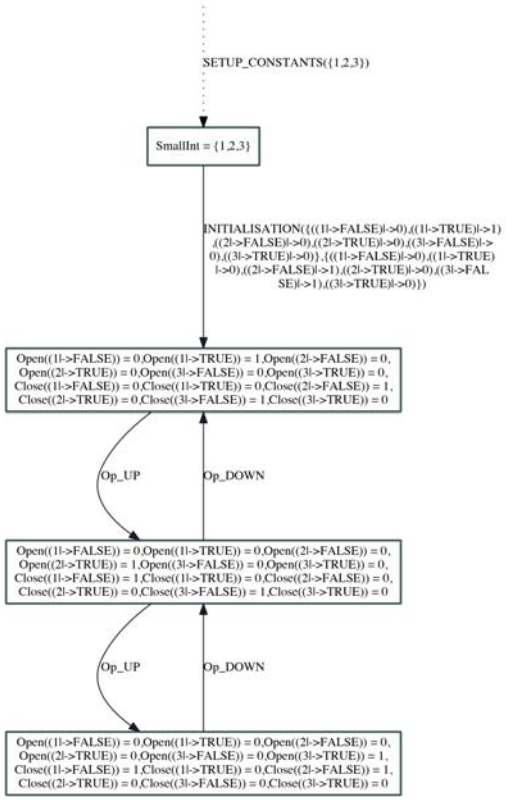
Lemma 3. Let a marking M be fired by a binding element $(t, b) \in BE$, and changed into a new marking M'. The image B machine must also recall the corresponding operation OP_t, because $M \xrightarrow{(t,b)} M' \Leftrightarrow VP' = [ACT_t\langle b^\beta\rangle]VP$.

The complete form of OP_UP is shown as follows:

```
OPERATIONS
  Op_UP=
  SELECT #(VAR_UP).(CDT_UP)
  THEN ANY VAR_UP
    WHERE CDT_UP
    THEN
      OpenDoor := Ms_Add(Ms_Less(OpenDoor,E_OpenDoor_UP,
          LiftDoor),E_UP_OpenDoor,LiftDoor) ||
      CloseDoor:= Ms_Add(Ms_Less(CloseDoor,E_CloseDoor_UP,
          LiftDoor),E_UP_CloseDoor,LiftDoor)
      END
  END;
```



(a) State space of CP-net



(b) State space of B-machine

Figure 7: Demo of transition transformation

Hence, for each transition and its image operation, the enabling conditions and occurrence rules are coherent. So the correctness of behaviour (dynamic) properties is maintained.

4.4.3 Model Checking

The coloured Petri net and the *B* language are both tool-supported formal method based on system state space. Since we have proved the static and dy-

namic properties of the translation by theorem analysis, we can use model checking as an auxiliary method of verification. Model checking has a mathematical representation of a system, and its result consists of a systematic exhaustive exploration of the mathematical model. There is already some research using model checking for verification of model transformations [Calegari and Szasz 2013]. Because the well known "state explosion" limitation exists, this approach can only be applied in a small system or used as an auxiliary method.

The comparison is shown in Fig. 7. The state space of CP-net is calculated by the CPN Tools, and the state space of B-machine is calculated by ProB. After initialisation, each model has 3 states , and 4 state changes. Their state space can be considered as the same, and this result reinforces the reliability of our translation method.

5 Demonstration for mapping non-hierarchical CP-net

This section presents two examples to illustrate the translation process of non-hierarchical CP-net. In order to allow the readers to have a better understanding of our method, only well-known cases are chosen in this paper, rather than some railway based cases.

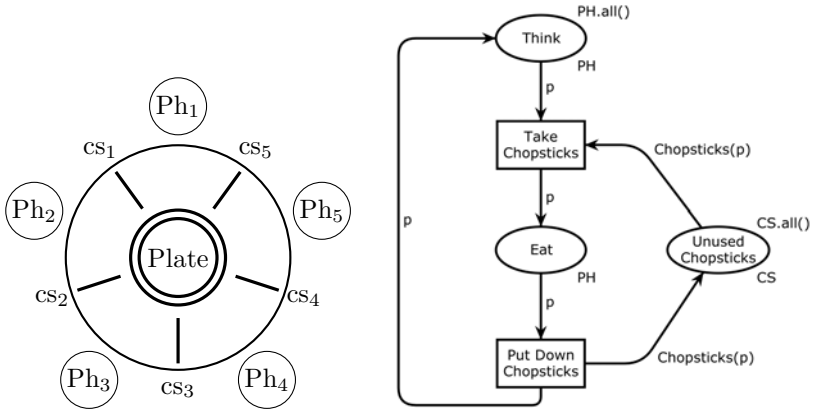
5.1 Dining philosophers problem

An example seen in Fig. 8(a) is the dining philosophers problem. In this case, five Chinese philosophers are having dinner at a round table. There are only one plate and five chopsticks on the table, and each chopstick is placed between two neighbouring philosophers. To eat the dish, each philosopher has to use two chopsticks next to him. Once a philosopher starts eating, his two neighbours have to wait until the chopsticks are unoccupied.

This system is modelled by CP-nets shown in Fig. 8(b), which could well express the concept of synchronization and concurrency. Philosophers are defined as a set of PH, and the chopsticks are defined as a set of CS. The $Chopsticks()$ is a mapping function which indicates the relationship between the philosophers and the chopsticks they can use.

The translation job could be roughly divided into three steps. First, the basic framework of the B machine should be built. Each place and each transition of the CP-net is mapped into the entries of variables and operations in the B-machine. Then, the colour information and user-defined CPN declarations are filled into the appropriate clauses. Finally, the detailed specifications of each arc, guard and transition function are complemented. For a clear mind, these steps are described separately. But in practice, they can be applied at the same time.

The corresponding B machine is shown in Listing 1. In this machine, we named the variables and the operations in the same way as when they are in the



(a) Problem describesion (b) CP-net of dining philosophers

```
val n = 5;
colset PH = index ph with 1..n;
colset CS = index cs with 1..n;
var p: PH;
fun Chopsticks(ph(i)) =
        1`cs(i) ++ 1`cs(if i=n then 1 else i+1);
```

Figure 8: Dining philosophers problem

places and transitions. The expressions starting with "E_" are the arc expressions. The source and destination of each arc are noted in an abbreviate form. For example, the arc `E_T_TkChs` means the arc from place *Think* to transition *Take_Chopsticks*. Similarly, the arc `E_PdChs_U` means the arc from transition *Put_Down_Chopsticks* to place *Unused*. The expression starting with "CDT_" is the enabling condition of each operation. Statements "skip" and "TRUE" are omitted in the machine, because for each generalized substitution S, it holds that $S||skip = S$, and for each predicate P that $P \wedge TRUE = P$.

In Listing 1, the symbol "`:`" stands for "belongs to", "`<<:`" is strict inclusion, "`:=`" is becomes equal to, "==" is definition, "<+" is overload a relation, "`INT`" is the set of integer numbers. "`|->`" is a maplet arrow.

In this machine, the user-defined function "Chopsticks(p)" is quite different from its original form in the CP-net, `fun Chopsticks(ph(i))=1`cs(i) ++ 1`cs(if i=n then 1 else i+1)`. That is because the conditional bounded choice "SELECT" will use this function in the predicate. But the original function is a "IF-THEN-ELSE-END" substitution which cannot be a predicate. So this function should be written as a formula, which can be proved by B language.

Listing 1: B-machine for dining philosophers

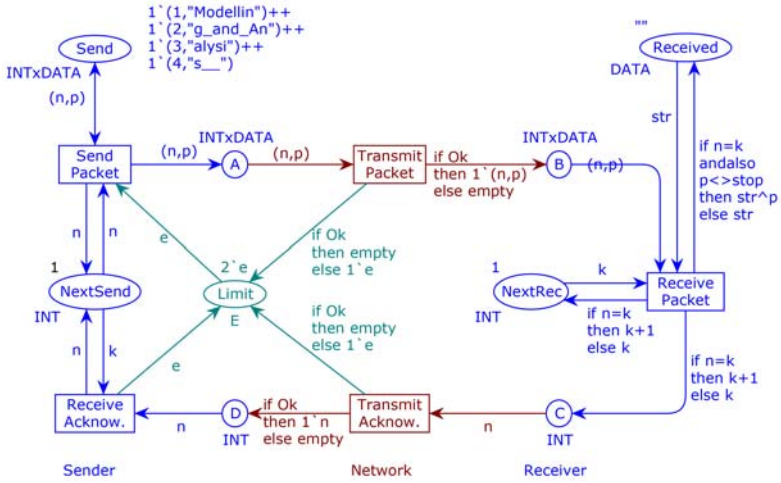
```
MACHINE dining_philosophers
CONSTANTS val_n, PH, CS
PROPERTIES
  val_n: INT & val_n = 5 &
  PH<<:INT & PH= 1..val_n & CS= 1..val_n
VARIABLES
  Think, Eat, Unused
INVARIANT
  Think : MS(PH) & Eat : MS(PH) & Unused : MS(CS)
INITIALISATION
  Think := Ms_Empty(PH) <+ PH*{1} ||
  Eat := Ms_Empty(PH) ||
  Unused := Ms_Empty(CS) <+ CS*{1}
DEFINITIONS
  "MultiSets.def"
  Chopsticks(pp)== {pp,((pp mod val_n)+1)};

  VAR_TkChs== pp;
    E_T_TkChs== (Ms_Empty(PH) <+ {pp|->1});
    E_U_TkChs== (Ms_Empty(CS) <+ Chopsticks(pp)*{1});
    E_TkChs_E== (Ms_Empty(PH) <+ {pp|->1});
  CDT_TkChs== pp:PH & Ms_Subset(Think,E_T_TkChs,PH) &
      Ms_Subset(Unused,E_U_TkChs,CS);

  VAR_PdChs== pp;
    E_E_PdChs== (Ms_Empty(PH) <+ {pp|->1});
    E_PdChs_T== (Ms_Empty(PH) <+ {pp|->1});
    E_PdChs_U== (Ms_Empty(CS) <+ Chopsticks(pp)*{1});
  CDT_PdChs== pp:PH & Ms_Subset(Eat,E_E_PdChs,PH)

OPERATIONS
  TakeChopsticks=
    SELECT #(VAR_TkChs).(CDT_TkChs)
    THEN ANY VAR_TkChs WHERE CDT_TkChs
      THEN Think:= Ms_Less(Think,E_T_TkChs,PH) ||
           Unused:= Ms_Less(Unused,E_U_TkChs,CS) ||
           Eat:= Ms_Add(Eat,E_TkChs_E,PH) END
    END;

  PutDownChopsticks=
    SELECT #(VAR_PdChs).(CDT_PdChs)
    THEN ANY VAR_PdChs WHERE CDT_PdChs
      THEN Think:=Ms_Add(Think,E_PdChs_T,PH) ||
           Unused:=Ms_Add(Unused,E_PdChs_U,CS) ||
           Eat:=Ms_Less(Eat,E_E_PdChs,PH) END
    END
END
```

The state of the dining_philosophers machine is shown in Tab. 4. It has 9 proof obligations, and could be automatically proved (PRa) by Atelier-B Ver.4.1.0.

AutoProved

	nPO	nPRi	nPRa	nUn	%Pr
Initialisation	3	0	3	0	100
Take_Chopsticks	3	0	3	0	100
Put_Down_Chopsticks	3	0	3	0	100
dining_philosophers	**9**	**0**	**9**	**0**	**100**

Table 4: Component status for dining_philosophers




```
colset INT = int;
colset DATA = string;
colset INTxDATA = product INT*DATA;
colset E = with e;
var n,k: INT;
var p,str: DATA;
var stop: "s__";
colset BOOL = bool;
var OK: BOOL;
```

Figure 9: CP-net of simple protocol

5.2 Simple protocol

In the previous example, all the arc inscriptions are quite simple, and only simple colour sets are involved. So the following example will demonstrate with compound colour sets and conditional arc inscriptions.

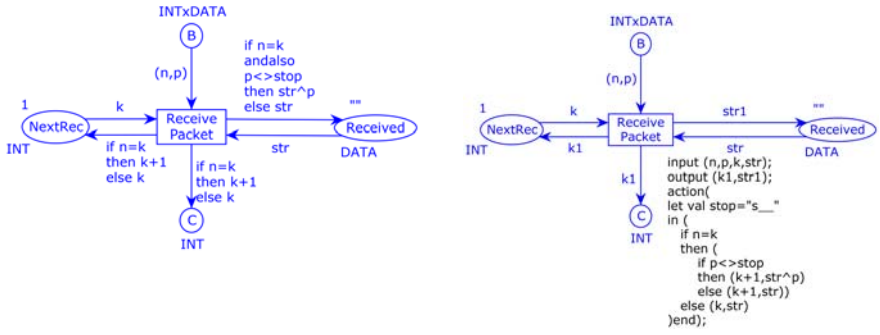
The example in Fig. 9 describes a simple protocol by which a sender can transfer a number of packets to a receiver. At most two packets could be sent at once. The communication process may lose packets, and different packets may overtake each other. Hence, it may be necessary to retransmit packets and to

ignore doublets and packets that are out of order.

The string colour set in the B machine becomes an enumerate set with all necessary strings and the combined string result could be a string sequence. The declarative part of the B machine is shown in Listing 2.

Listing 2: B-machine declarative part of simple protocol

```
MACHINE SimpleProtocol
SETS DATA={Modellin, g_and_An, alysi, s__}; EV={ee}
CONSTANTS
  INTxDATA,val_stop,DataLst
PROPERTIES
  INTxDATA = INTEGER*DATA & val_stop = s__ & DataLst =
      seq(DATA)
VARIABLES
  Send, Limit, Received, NextSend, NextRec, AA, BB, CC,
      DD
INVARIANT
  Send: MS(INTxDATA) & Limit: MS(EV) & Received: MS(
      DataLst) &
  NextSend: MS(INTEGER) & NextRec: MS(INTEGER) &
  AA: MS(INTxDATA) & BB: MS(INTxDATA) &
  CC: MS(INTEGER) & DD: MS(INTEGER)
INITIALISATION
  Send := Ms_Empty(INTxDATA) <+ {(1|->Modellin)|->1,(2|->
      g_and_An)|->1,(3|->alysi)|->1,(4|->s__)|->1} ||
  Limit := Ms_Empty(EV) <+ {ee|->2} ||
  Received := Ms_Empty(DataLst) <+ {[]|->1} ||
  NextSend := Ms_Empty(INTEGER) <+ {1|->1} ||
  NextRec := Ms_Empty(INTEGER) <+ {1|->1} ||
  AA := Ms_Empty(INTxDATA) ||
  BB := Ms_Empty(INTxDATA) ||
  CC := Ms_Empty(INTEGER) ||
  DD := Ms_Empty(INTEGER)
```



(a) Original transition (b) New transition

Figure 10: Demo of transition transformation

There are five transitions in the model. The transitions *TransmitPacket*, *TransmitAcknow* and *ReceivePacket* have conditional output arcs. In our framework the arcs are defined in the Definition clause of the machine, but the conditional expressions cannot be applied in Definitions. So a transformation of transition is needed to generate a new formalism, which has the same function but simpler arcs.

The transformation ensures consistency of original transition but has a different notation. All the arc inscriptions will be concentrated inside the transition function, and maintain conciseness of all the arc expressions. A transformation demo of transition *ReceivePacket* is shown in Fig. 10.

After transforming the three transitions into the simple form, we can continue to translate the CP-net, and some of the operations are shown in Listing 3, where `TrPck` is short for "TransmitPacket", `RcPck` is short for "ReceivePacket". The state of this machine is shown in Tab. 5.

AutoProved

	nPO	nPRi	nPRa	nUn	%Pr
Initialisation	6	0	6	0	100
Op_SendPacket	2	0	2	0	100
Op_ReceiveAcknow	3	0	3	0	100
Op_TransmitPacket	2	0	2	0	100
Op_TransmitAcknow	2	0	2	0	100
Op_ReceivePacket	4	0	4	0	100
SimpleProtocol	**19**	**0**	**19**	**0**	**100**

Table 5: Component status for SimpleProtocol

6 Conclusion

This paper discusses the combination usages of different modelling languages in the context of safety critical system. The method presented here is a model transformation from coloured Petri net models to abstract *B* machines while preserving the same modelling requirements. The strong motivation of such an approach is to bridge the gap of critical tasks, from a strong requirement analysis towards a valid implementation on a real system. In the design stage, this approach may assist the designers on the way from analysis to implementation.

Listing 3: Part of operations of simple protocol

```
DEFINITIONS
  "MultiSets.def"
  // TransmitPacket
  VAR_TrPck== nn,pp;
    E_A_TrPck== (Ms_Empty(INTxDATA) <+ {(nn|->pp)|->1});
    E_TrPck_B== (Ms_Empty(INTxDATA) <+ {(nn|->pp)|->1});
    E_TrPck_L== (Ms_Empty(EV) <+ {ee|->1});
  CDT_TrPck== (nn:INTEGER & pp:DATA & Ms_Subset(AA,
      E_A_TrPck, INTxDATA));

  // ReceivePacket
  VAR_RcPck== nn,pp,str,kk;
    E_B_RcPck== (Ms_Empty(INTxDATA) <+ {(nn|->pp)|->1});
    E_R_RcPck== (Ms_Empty(DataLst) <+ {str|->1});
    E_RcPck_R(str1)== (Ms_Empty(DataLst) <+ {str1|->1});
    E_N_RcPck== (Ms_Empty(INTEGER) <+ {kk|->1});
    E_RcPck_N(kk1)== (Ms_Empty(INTEGER) <+ {kk1|->1});
    E_RcPck_C(kk1)== (Ms_Empty(INTEGER) <+ {kk1|->1});
  CDT_RcPck== (nn:INTEGER & pp:DATA & str:DataLst & kk:
      INTEGER & Ms_Subset(BB, E_B_RcPck, INTxDATA) &
      Ms_Subset(NextRec, E_N_RcPck, INTEGER) & Ms_Subset(
      Received, E_R_RcPck, DataLst))

OPERATIONS
  Op_TransmitPacket= SELECT #(VAR_TrPck).(CDT_TrPck)
    THEN ANY VAR_TrPck,OK
      WHERE CDT_TrPck & OK:BOOL
      THEN AA:= Ms_Less(AA,E_A_TrPck,INTxDATA) ||
           IF OK=TRUE
           THEN BB:= Ms_Add(BB,E_TrPck_B,INTxDATA)
           ELSE Limit:= Ms_Add(Limit,E_TrPck_L,EV) END
               END END;

  Op_ReceivePacket= SELECT #(VAR_RcPck).(CDT_RcPck)
   THEN ANY VAR_RcPck WHERE CDT_RcPck
    THEN
      BB:= Ms_Less(BB,E_B_RcPck,Colset_INTxDATA) ||
      IF nn=kk
      THEN
        IF pp/=val_stop
        THEN Received := Ms_Add(Ms_Less(Received,
            E_R_RcPck,DataLst),E_RcPck_R(str<-pp),DataLst
            )
        ELSE Received := Ms_Add(Ms_Less(Received,
            E_R_RcPck,DataLst),E_RcPck_R(str),DataLst)
            END ||
        NextRec := Ms_Add(Ms_Less(NextRec,E_N_RcPck,
            INTEGER),E_RcPck_N(kk+1),INTEGER) ||
        CC := Ms_Add(CC,E_RcPck_C(kk+1),INTEGER)
      ELSE Received := Ms_Add(Ms_Less(Received,E_R_RcPck,
          DataLst),E_RcPck_R(str),DataLst) ||
        NextRec := Ms_Add(Ms_Less(NextRec,E_N_RcPck,
            INTEGER),E_RcPck_N(kk),INTEGER) ||
        CC := Ms_Add(CC,E_RcPck_C(kk),INTEGER)
        END END END
```

As the coloured Petri nets have an easy-to-understand graphic notation, so they may be more attractive for designers to use than text-based methods. But there is a switch point when the implementation consideration is introduced. The *B* method has been designed for software development but it is not user-friendly. Still, the *B* proved system is accepted as a strong safety proof in the French railway context. So, our approach cannot only help with the implementation of coloured Petri net specifications, but it also can integrate information from different parts, such as enable people from critical tasks to communicate with the same safety requirement. Because the coloured Petri net is mainly used by automation engineers or electromechanics engineers, while the *B* method is rather used by software engineers, such a transformation can build a bridge between them. In engineering practice, if we want to validate safety at a system level, this transformation will be a strong contribution.

The model transformation that is introduced in this paper is designed for non-hierarchical coloured Petri nets. First, the fundamental definitions of both non-hierarchical CP-nets and abstract *B* machines are presented, so that a systematic translation can be introduced. Second, the detailed mapping process of non-hierarchical CP-nets and abstract machine notations is explained in three parts: 1. a model framework for mapping the Petri net's "Place-transition" structure; 2. the multi-set mechanism in *B* formalism; 3. different colour sets in abstract machine notation. Then, a global design is presented for proving the equivalent of this model transformation. We use a small example to illustrate that both static state and dynamic behaviour are maintained between the CP-net model and the transformed *B* machine. Finally, two case studies are presented, which show our transformation is compatible with various types of colour sets and the transformed *B* machines can be automatically proved by Atelier B.

One of the limitations of this paper is that the colour-set "list" is not so perfect. Because it is already a concrete programming data type in CP-net, it is difficult to abstract all its function with set theory and first order logic. Another limitation is that the current translation method does not support timed CP-nets, which are associated with a timed multi-set and new behaviour rules. Considering the timetable and time constrained protocol, it could be an interesting research task. Note that the *B* machine dealing with the time concept still needs some scientific work. However, further research will be conducted considering more practical aspects.

Acknowledgements

The work is funded by the French National Research Agency (ANR) in the context of the project PERFECT (Performing Enhanced Rail Formal Engineering Constraints Traceability). This project is also supported by the French i-Trans

competitive pole. The Ph.D. of Pengfei SUN is funded by the China Scholarship Council (CSC).

References

[Abrial 1996] Abrial, J.-R.: The B-book: Assigning Programs to Meanings; Cambridge University Press, New York, NY, USA, 1996.

[Antoni 2009] Antoni, M.: "Formal validation method for computerized railway interlocking systems"; Computers Industrial Engineering, 2009. CIE 2009. International Conference on; 1532–1541; 2009.

[Antoni 2012] Antoni, M.: "Méthode de validation formelle d'un poste d'aiguillage informatique"; Recherche Transports Sécurité; 28 (2012), 2, 101–118.

[Attiogbe 2009] Attiogbe, C.: "Semantic embedding of petri nets into event-B"; Integration of Model-based Formal Methods Tools (IM_FMT@ IFM'2009); 2009.

[Behm et al. 1999] Behm, P., Benoit, P., Faivre, A., Meynadier, J.-M.: "Météor: A successful application of b in a large project"; J. Wing, J. Woodcock, J. Davies, eds., FM99 Formal Methods; volume 1708 of Lecture Notes in Computer Science; 369–387; Springer Berlin Heidelberg, 1999.

[Bon 2000] Bon, P.: Du cahier des charges aux spécifications formelles : une méthode basée sur les réseaux de Petri de haut niveau; Ph.D. thesis; Université des Sciences et Techniques de Lille (2000).

[Bon and Dutilleul 2013] Bon, P., Dutilleul, S. C.: "From a solution model to a B model for verification of safety properties."; J. UCS; 19 (2013), 1, 2–24.

[Boulanger 2013a] Boulanger, J.-L.: Formal Methods: Industrial Use from Model to the Code; ISTE; Wiley, 2013a.

[Boulanger 2013b] Boulanger, J.-L.: Industrial Use of Formal Methods: Formal Verification; ISTE; Wiley, 2013b.

[Boulanger 2014] Boulanger, J.-L.: Formal Methods Applied to Complex Systems; John Wiley & Sons, Inc., 2014.

[Buchheit et al. 2011] Buchheit, G., Malassé, O., Brinzei, N., Lalouette, J., Walter, M., et al.: "Évaluation des performances d'un axe ferroviaire en fonction des caractéristiques fiabilistes de ses systèmes de signalisations"; 9ème Congrès International Pluridisciplinaire Qualité et Sûreté de Fonctionnement, Qualita'2011; (2011).

[Calegari and Szasz 2013] Calegari, D., Szasz, N.: "Verification of model transformations: A survey of the state-of-the-art"; Electronic Notes in Theoretical Computer Science; 292 (2013), 0, 5 – 25; proceedings of the XXXVIII Latin American Conference in Informatics (CLEI).

[Erbin and Soulas 2003] Erbin, J., Soulas, C.: "Twenty years of experiences with driverless metros in france"; VWT19; (2003), 1–33.

[Jensen 1987] Jensen, K.: Coloured Petri Nets; Springer Berlin Heidelberg, 1987.

[Jensen and Kristensen 2009] Jensen, K., Kristensen, L. M.: Coloured Petri nets: modelling and validation of concurrent systems; Springer, 2009.

[Korečko et al. 2007] Korečko, Š., Hudák, Š., Šimoňák, S.: "Petri net-like treatment of B-machine behaviour"; Journal of Information, Control and Management Systems; 5 (2007), 2.1.

[Korečko and Sobota 2014] Korečko, Š., Sobota, B.: "Petri nets to B-language transformation in software development"; Acta Polytechnica Hungarica; 11 (2014), 6.

[Lalouette et al. 2010] Lalouette, J., Caron, R., Scherb, F., Brinzei, N., Aubry, J.-F., Malassé, O., et al.: "Évaluation des performances du système de signalisation ferroviaire européen superpose au système français, en présence de défaillances"; 17e Congrès de Maîtrise des Risques et de Sûreté de Fonctionnement, Lambda-Mu'2010; (2010).

[Lecomte 2008] Lecomte, T.: "Safe and Reliable Metro Platform Screen Doors Control/Command Systems"; FM 2008: Formal Methods; 430–434; Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Milner 1997] Milner, R.: The definition of standard ML: revised; MIT press, 1997.

[Patin et al. 2006] Patin, F., Pouzancre, G., Servat, T.: "A formal approach in the implementation of a safety system for automatic control of platform doors"; 4th AFIS Conference on System Engineering; 2006.

[Sun et al. 2014] Sun, P., Collart-Dutilleul, S., Bon, P.: "Formal modeling methodology of french railway interlocking system via HCPN"; COMPRAIL 14, International conference on Railway Engineering Design and Optimization; 2014.