

Concordia University  
Department of Computer Science and Software  
Engineering

**SOEN 331 - S and U**  
**Introduction to Formal Methods**  
**for Software Engineering**

**Assignment 2 - Solutions**  
The Object-Z specification language  
**Team 19 - Section U**

<b>Samuel Boaknin</b>	<b>Ryan Leyland</b>	<b>Saleha Tariq</b>
-----------------------	---------------------	---------------------

40009692

40015165

40006997

**Meng Susana Ung**

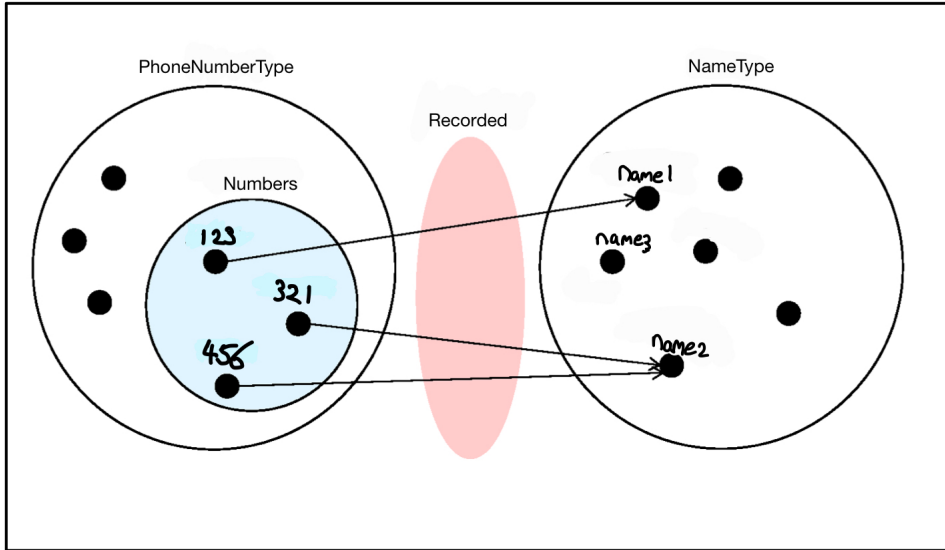
40099729

Due Date: March 11, 2021

# 1 Contact management

## 1.1 State

1. This is our diagram for the state of the system:



2. The formal definition of *numbers* is as follows:

$$numbers : \mathbb{P} \text{ PhoneNumberType}$$

3. The relation *recorded* must indeed be captured by a function. A function, by definition, enforces that each element of a source set (that is, the set in the domain) is mapped to exactly one element of a target set (a set of elements in the range). Part of the requirements for the system is that people may not share phone numbers. By making *recorded* a function, and setting the *numbers* set as a subset of the domain, the requirement of a phone number being associated to a single person is trivially met. If this is not done, an additional series of specifications must be defined such that this requirement is observed *at best*, and at worst, there is a severe risk of failing to meet the defined requirements. Thus, it is at least convenient, if not necessary that *recorded* is a function where *numbers* is a subset of the domain.

4. The domain of *recorded* is *numbers*, and its codomain is *NameType*.
5. The function *recorded* is a partial function, because partial functions, by definition, do not map all elements in the domain to an element in the co-domain. Specifically, *recorded* doesn't relate every element of *PhoneNumberType* to *NameType*.
6. The function *recorded* will not be either injective, surjective, or bijective. It cannot be injective because an injective function, by definition, is a function where one element in the domain is mapped to at most one element in the co-domain, but *recorded* allows for multiple phone numbers to be associated with one person, so multiple parts of the domain could be associated with an element of the co-domain. It cannot be surjective as a surjective function, by definition, is a function where each element in the co-domain has at least one element in the domain mapped to it. In *recorded*, there are elements contained in the co-domain of *recorded* that are not in the range of *recorded* therefore there exists elements in the codomain which are not mapped to by any element of the domain. It cannot be bijective because a bijective function is both injective and surjective and it has already been proven that this function is neither. Therefore, *recorded* must be a general function because every element in the domain is mapped to one element of the co-domain, but more than one element of the domain can be mapped to the same element of the co-domain.
7. The formal definition of *recorded* is as follows:

$$\textit{recorded} : \textit{PhoneNumberType} \rightarrow \textit{NameType}$$

## 1.2 Class Contacts

<i>Contacts</i>
$\uparrow (MakeNewContact, AddNumber, SearchForNumber, DeleteNumber)$
$numbers : \mathbb{P}PhoneNumberType$ $recorded : PhoneNumberType \leftrightarrow NameType$
$numbers = dom\ recorded$
<i>INIT</i>
$recorded = \emptyset$
<i>MakeNewContactOK</i>
$\Delta(recorded)$ $number? : PhoneNumberType$ $name? : NameType$
$number? \notin numbers$ $name? \notin ran\ recorded$ $recorded' = recorded \cup \{number? \mapsto name?\}$
<i>AddNumberOK</i>
$\Delta(recorded)$ $number? : PhoneNumberType$ $name? : NameType$
$number? \notin numbers$ $name? \in ran\ recorded$ $recorded' = recorded \cup \{number? \mapsto name?\}$
<i>SearchForNumberOK</i>
$\exists Contacts$ $name? : NameType$ $numbers! : \mathbb{P}number$
$numbers! = \{n : number \mid recorded(n) = name?\}$
...

*Contacts/cont.*

...

*DeleteNumberOK*

$\Delta(\text{recorded})$

$\text{number?} : \text{PhoneNumberType}$

$\text{name} : \text{NameType}$

$\text{number?} \in \text{numbers}$

$\text{name} = \text{recorded}(\text{number?})$

$\text{recorded}' = \{\text{number?}\} \triangleleft \text{recorded}$

*NameUnknown*

$\text{name?} : \text{NameType}$

$\text{response!} : \text{Message}$

$\text{name?} \notin \text{ran recorded}$

$\text{response!} = \text{'Name unknown'}$

*NumberUnknown*

$\text{number?} : \text{PhoneNumberType}$

$\text{response!} : \text{Message}$

$\text{number?} \notin \text{dom recorded}$

$\text{response!} = \text{'Number unknown'}$

*NameExists*

$\text{name?} : \text{NameType}$

$\text{response!} : \text{Message}$

$\text{name?} \in \text{ran recorded}$

$\text{response!} = \text{'Name already exists'}$

*NumberExists*

$\text{number?} : \text{PhoneNumberType}$

$\text{response!} : \text{Message}$

$\text{number?} \in \text{dom recorded}$

$\text{response!} = \text{'Number already exists'}$

*ContactsEmpty*

$\text{response!} : \text{Message}$

$\text{response!} = \text{'Contacts is empty'}$

...

*Contacts/cont.*

...

*Success*

*response! : Message*

*response! = 'Success'*

$MakeNewContact \hat{=} (MakeNewContactOK \wedge Success) \oplus (NameExists \vee NumberExists)$

$AddNumber \hat{=} (AddNumberOK \wedge Success) \oplus (NameUnknown \vee NumberExists \vee ContactsEmpty)$

$SearchForNumber \hat{=} (SearchForNumberOK \wedge Success) \oplus (NameUnknown \vee ContactsEmpty)$

$DeleteNumber \hat{=} (DeleteNumberOK \wedge Success) \oplus (NumberUnknown \vee ContactsEmpty)$

### 1.3 Class Contacts2

*Contacts2*

$\uparrow (MakeNewContact, AddNumber, SearchForNumber, DeleteNumber, SearchForPerson)$

*Contacts*

*SearchForPersonOK*

$\exists Contacts2$

$number? : PhoneNumberType$

$name! : NameType$

$number? \in numbers$

$name! = recorded(number?)$

$SearchForPerson \hat{=} (SearchForPersonOK \wedge Success) \oplus (NumberUnknown \vee ContactsEmpty)$