SOFTWARE VERIFICATION RESEARCH CENTRE

DEPARTMENT OF COMPUTER SCIENCE

THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 95-55

Formal Verification of Object-Z Specifications

Graeme Smith

December 1995

Phone: +61 7 3365 1003
Fax: +61 7 3365 1533

Note: Most SVRC technical reports are available via anonymous ftp, from `ftp.cs.uq.edu.au` in the directory `/pub/SVRC/techreports`.

Formal Verification of Object-Z Specifications

Graeme Smith
Software Verification Research Centre
Department of Computer Science
University of Queensland, Australia

Abstract

This paper suggests extensions to the existing logic for Object-Z in order to facilitate the process of formal verification. Tactics are introduced to simplify large, complex proofs and an additional inference rule is introduced to control aliasing. The inference rule is based on the notion that objects within a specified system are not aliased with objects outside that system, i.e. objects of other systems with which the specified system may be composed. The extensions are illustrated through a mobile phone system case study.

1 Introduction

Formally verifying a property of a system specification involves showing that the property can be derived from the specification using the rules of a mathematically-sound logic. A logic for the object-oriented formal specification language Object-Z[4, 7, 8] is presented in [9][1]. A denotational semantics of Object-Z with respect to which this logic could be proved sound is outlined in [5]. The purpose of this paper is to extend the logic for Object-Z in order to facilitate the process of formal verification. This is done in two ways.

- Firstly, a number of *tactics* are introduced. Tactics are reusable sequences of proof steps used to facilitate and simplify large, complex proofs.
- Secondly, a simple notion of *alias control* is introduced. Aliasing occurs when there is more than one reference to a particular object in a specification[2]. This needs to be limited in order to practically reason about a system.

Both extensions are illustrated through a case study based on the mobile phone system specification in [8]. Section 2 introduces Object-Z and its logic. Section 3 outlines the case study and suggests an implicit property of the specification. Informal justifications for parts of the proof of this property are given. These informal justifications are used to motivate the tactics presented in Section 4 and the rule for alias control presented in Section 5.

[1]Extensions and revisions to this logic can be found in [10].
[2]A typical Object-Z specification describes a system as a collection of interacting objects.

2 Object-Z

Object-Z is an extension to the formal specification language Z[1, 11, 6][3] which facilitates specification in an object-oriented style. Enhanced structuring is provided through the addition to Z of *classes*. A class encapsulates types, constants and variables describing the state of particular objects in the system with schemas describing their initial states and the operations which they can undergo. Classes can be defined as incremental modifications to existing classes by inheriting their definitions.

A class is not merely a syntactic extension but also defines a type whose instances are references to objects of the class. For example, given a class A the declaration $a : A$ declares a reference to an object of class A and $a : \downarrow A$ declares a reference to an object of class A or any class derived from A by inheritance[4]. The constants and variables of a referenced object can be accessed using the dot notation, e.g. $a.x$ denotes the constant or variable x of the object referenced by a. The dot notation is also used to specify that an object is in its initial state, e.g. $a.\textsc{Init}$, and to apply an operation to an object, e.g. $a.Op$ where Op is the name of an operation of A.

An Object-Z class is represented syntactically as a named box in which there may be local type and constant definitions, at most one state schema and associated initial state schema, and zero or more operations. In addition to these explicit definitions, a class may *inherit* the definition of one or more other classes. It may also have generic parameters, a visibility list restricting the way objects of the class may be used, and a history invariant for capturing liveness properties; these, however, are not discussed in this paper.

ClassName

inherited class designators
local type and constant definitions
state schema
initial state schema
operations

A class designator consists of a class name and possibly a rename list. The rename list allows constants, state variables and operations to be renamed. For example, given a class A with state variables x and y and operations Op_1 and Op_2, the following is a class designator which renames y to z and Op_2 to Op_3.

$$A[z/y, Op_3/Op_2]$$

When a class is inherited, its local types and constants are implicitly available in the inheriting class. Any types or constants with the same name occurring in both the inherited and the inheriting class are semantically identified and hence must have compatible definitions.

The inherited class' state schema and initial state schema are implicitly conjoined with those declared explicitly in the inheriting class. The inherited class' operations are implicitly available except where the name of an operation in the inheriting class is the same as that of an operation of the inherited class. In this case, the inherited operation is implicitly conjoined with that in the inheriting class.

[3]This paper assumes a familiarity with Z. Readers unfamiliar with Z may wish to consult [11] or [6].

[4]Object references to a member of a class in an arbitrary collection of classes[2] or to objects "contained" in a class[3] can also be declared. These notions will not be discussed in this paper.

The local type and constant definitions of a class have the same syntax as global type and constant definitions in Z. Their scope, however, is limited to the class in which they are declared. A constant is associated with a fixed value which cannot be changed by any operations of the class. However, the value of constants may differ for different objects of the class.

The state schema is nameless and has its declarations partitioned by a Δ into *primary* and *secondary* variables. Secondary variables are implicitly available for change in any operation and are usually defined in terms of the primary variables. For example, the state schema of a 'square' class may have *side* as a primary variable and *area* and *perimeter* as secondary variables.

$$
\begin{array}{|l}
side : \mathbb{N} \\
\Delta \\
area : \mathbb{N} \\
perimeter : \mathbb{N} \\
\hline
area = side * side \\
perimeter = 4 * side
\end{array}
$$

The initial state schema is named with the keyword INIT and has only a predicate part. It implicitly includes both the declarations and the predicates of the state schema.

The operations are defined either as *operation schemas* or *operation expressions*. They are interpreted in the class' local environment (i.e. the global environment of the specification enriched with the local type and constant definitions) enriched with the declarations and predicates of the state schema in both primed and unprimed form. Therefore, if $a : A$ is declared in the state schema of a class, $a.Op$ is a valid operation definition when Op is an operation of A.

An operation schema extends the notion of a schema in Z by adding to it a *delta-list*. The delta-list is a list of the primary variables which the operation may change when it is applied to an object of the class; all other primary variables remain unchanged[5]. When two or more operations are combined to define a new operation, however, their delta-lists are united so that the new operation may change any variable which any of its constituent operations could have changed. In this way, delta-lists enable a more flexible calculus for combining operations than would be allowed by Z schemas alone.

For example, given the operations Inc_x and Inc_y which increment the variables x and y respectively,

$$
\begin{array}{|l}
\multicolumn{1}{l}{Inc\_x} \\
\hline
\Delta(x) \\
\hline
x' = x + 1
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{Inc\_y} \\
\hline
\Delta(y) \\
\hline
y' = y + 1
\end{array}
$$

the operation expression $Inc\_x \wedge Inc\_y$ is equivalent to the following operation schema.

[5] When no delta-list appears in an operation, this is equivalent to having an empty delta-list, i.e. no primary variables can change.

$$
\begin{array}{|l}
\hline
\textit{Inc\_x} \wedge \textit{Inc\_y} \\
\Delta(x, y) \\
\hline
x' = x + 1 \\
y' = y + 1 \\
\hline
\end{array}
$$

Apart from conjunction $\wedge$, other operation operators are the *parallel operator* $\|$, the *choice operator* $[\!]$, the *enrichment operator* $\bullet$, hiding and renaming[6]. Distributed forms of $\wedge$, $\|$ and $[\!]$ also exist.

The parallel operator $\|$ is a binary operator introduced into Object-Z to allow specification of inter-object communication. The operator identifies and equates input variables in either operation with output variables in the other operation having the same basename, i.e. apart from the ? or !. The identified input variables are hidden in the resulting operation; the output variables are not hidden and so may be equated with other input variables in subsequent parallel compositions.

The choice operator $[\!]$ is a binary operator which allows the specification of nondeterministic choice between operations. The meaning of $Op_1 \ [\!] \ Op_2$ is that either operation Op_1 occurs or operation Op_2 occurs but not both. For example, $\textit{Inc\_x} \ [\!] \ \textit{Inc\_y}$ is equivalent to the following operation schema.

$$
\begin{array}{|l}
\hline
\textit{Inc\_x} \ [\!] \ \textit{Inc\_y} \\
\Delta(x, y) \\
\hline
(x' = x + 1 \wedge y' = y \\
\vee \\
y' = y + 1 \wedge x' = x) \\
\hline
\end{array}
$$

The enrichment operator $\bullet$ allows operations to be interpreted within the class' local environment enriched with the declarations and predicates of another operation or schema. It is particularly useful when specifying operations occurring on particular objects in a set of objects: references to the selected objects appear in the declarations of the first operation and can be used in the definition of the second. For example, given the declaration $s : \mathbb{P}\, A$ in the state schema of a class, the operation which involves the parallel composition of two distinct objects referenced from s performing Op (an operation of A) is as follows.

$$[\, a_1, a_2 : s \mid a_1 \neq a_2 \,] \bullet a_1.Op \; || \; a_2.Op$$

Hiding and renaming are defined as in Z. Since the primary and secondary variables (in both primed and unprimed form) are only available to an operation definition (and not part of it), they cannot be hidden or renamed.

2.1 A logic for Object-Z

A logic for reasoning about Object-Z specifications is presented in [9]. It extends the logic presented for Z in the Z Base Standard[1]. The logic is a Gentzen-style sequent calculus. Axioms and theorems are expressed using *sequents* and further theorems are derived by the application of *inference rules*. The expression of these rules is facilitated by the use of *meta-functions* defined outside the logic.

[6]Object-Z also has a sequential composition operator which will not be discussed in this paper.

2.1.1 Sequents

Sequents are of the form

$$A :: \quad d \mid \Psi \vdash \Phi$$

where A is a class name, d is a list of declarations and Ψ and Φ are lists of predicates. The declarations d and predicates Ψ are referred to as the *antecedent* and the predicates Φ as the *consequent*. A sequent is *valid* if at least one of the predicates in Φ is true in the local environment of A enriched by the declarations d and satisfying all predicates in Ψ.

For example, if the specification contains the following class

$$\begin{array}{|l}
\hline
\textbf{\textit{A}} \\
\quad \begin{array}{|l} x : \mathbb{N} \\ \hline x < 10 \end{array} \\
\quad \vdots \\
\hline
\end{array}$$

then the following are valid sequents.

$$\begin{array}{ll} A :: & \vdash x \in \mathbb{N} \\ A :: & \vdash x < 10 \end{array}$$

Each of the class name, antecedent and consequent of a sequent are optional. A sequent without a class name is interpreted in the global environment of the specification. An empty antecedent is equivalent to the empty declaration and the single predicate *true*. An empty consequent is equivalent to the single predicate *false*. (In effect, predicates in Ψ are conjoined whereas predicates in Φ are disjoined.)

2.1.2 Inference Rules

Inference rules for manipulating sequents are of the form

$$\frac{\textit{premisses}}{\textit{conclusion}} \quad (\ \textit{name}\ )\ [\ \textit{proviso}\ ]$$

where the *premisses* are zero or more sequents and the *conclusion* is a single sequent. The *name* identifies the rule and the *proviso* is a predicate which must be true in the global environment of the specification for the rule to be applicable. An inference rule is *sound* if, whenever the proviso is satisfied and the premisses are valid, the conclusion is also valid.

As an example, consider the following rule for logical disjunction.

$$\frac{p_1 \vdash \qquad p_2 \vdash}{p_1 \lor p_2 \vdash} \quad (\lor\vdash)$$

Such a rule (with empty consequents) is intended to be used in combination with a "rule-lifting" meta-theorem. This meta-theorem allows a common declaration to be added to every antecedent in a rule or a common predicate to be added to either every antecedent or every consequent. Therefore, given that the above rule is sound, so is the following rule which is derived by adding the predicate q to each consequent.

$$\frac{p_1 \vdash q \qquad p_2 \vdash q}{p_1 \vee p_2 \vdash q}$$

This rule states that if we can deduce the predicate q from the predicates p_1 and p_2 then we can deduce q from the predicate $p_1 \vee p_2$.

So that such inference rules can be used in the context of a class, the following meta-theorem is introduced[7].

If the rule $$\frac{d_1 \mid \Psi_1 \vdash \Phi_1}{d_2 \mid \Psi_2 \vdash \Phi_2} \; [\, p \,]$$ is sound,

then the rule $$\frac{A :: \quad d_1 \mid \Psi_1 \vdash \Phi_1}{A :: \quad d_2 \mid \Psi_2 \vdash \Phi_2} \; [\, p \,]$$ is also sound.

Further inference rules can be derived via application of a "semantic-equivalence-lifting" meta-theorem. This meta-theorem allows predicates and declarations within a sequent to be replaced by equivalent ones. That is, given that $p \equiv q$ and $d_1 \equiv d_2$ then the following are sound inference rules[8].

$$\frac{d_2 \mid q \vdash}{d_1 \mid p \vdash} \qquad\qquad \frac{d_2 \vdash q}{d_1 \vdash p}$$

2.1.3 Meta-Functions and Operators

The logic makes use of meta-functions to facilitate the expression of rules. For example, the meta-function α returns the names declared in its argument. If the argument is a schema, α returns the names of the variables declared in the signature of the schema. If its argument is a class, α returns the names of the local types, constants and state variables of the class. The meta-functions α_1 and α_2 return the set of primary and secondary state variables of a class respectively. The meta-function ϕ returns the free variables of its argument which may be an expression, predicate, declaration or schema.

The logic also has two meta-substitution operators. The first $\odot$ allows a binding to be substituted into an expression, predicate or declaration. When a binding is substituted into an expression, the free variables in the expression which are in the domain of the binding are replaced by the corresponding expressions in the range of the binding. For example, if the domain of the binding b contains x and y then:

$b \odot (x \in y)$ evaluates to $b.x \in b.y$

[7]This is a revised version of the meta-theorem in [9] which unnecessarily introduced class context into the proviso.

[8]$\equiv$ is meta-equivalence.

When substituting into a declaration the substitution is made into the types only, i.e. for b as above:

$b \odot (x : y)$ evaluates to $x : b.y$

This meta-substitution operator is used to define renaming. For example, to rename x to y in the predicate p the following notation is adopted.

$\langle\!| \; x \rightsquigarrow y \; |\!\rangle \odot p$

To simplify notation, it is assumed that a rename list is of the form $x_1 \rightsquigarrow s_1, \ldots, x_n \rightsquigarrow s_n$. Thus, the predicate of the schema $S[\lambda]$ is $\langle\!| \; \lambda \; |\!\rangle \odot S$.

The second meta-substitution operator $\circledast$ enables the substitution of the types, attributes and primed variables of an object into a predicate. It is defined in terms of the meta-substitution operator $\odot$ as follows.

$$
\begin{array}{l}
a \circledast p = b \odot p \\
\text{where } a \in A \\
\qquad \{x_1, \ldots, x_n\} = \alpha(A) \\
\qquad \{x_k, \ldots, x_n\} = \alpha_1(A) \cup \alpha_2(A) \\
\qquad b = \langle\!| \; x_1 \rightsquigarrow a.x_1, \ldots, x_n \rightsquigarrow a.x_n, x_k' \rightsquigarrow a.x_k', \ldots, x_n' \rightsquigarrow a.x_n' \; |\!\rangle
\end{array}
$$

Similarly, $\circledast$ is defined for substitution of the types, attributes and primed variables of an object into expressions and declarations.

3 Mobile Phone System

An Object-Z specification of a mobile phone system is presented in [8]. The specification describes a system consisting of a set of mobiles and a set of cells. The role of the cells is to transmit signals to, and receive signals from, mobiles in their designated region. These signals are passed along control channels (e.g. for paging a mobile) as well as voice channels of predetermined frequencies.

Mobiles are modelled by a class *Mobile* whose details will not be elaborated upon here. A cell has a frequency allocator component. This is specified by a class *FreqAllocator*. This class models a cell's set of predetermined allocatable frequencies (the local constant *freqs*) and its associated tuned mobiles and their allocated frequencies (the state variable *alloc*).

$$
\begin{array}{|l}
\hline
\textit{FreqAllocator} \\
\quad freqs : \mathbb{F}\, Frequency \\
\hline
\quad alloc : Mobile \rightarrowtail\!\!\!\!\rightarrow Frequency \\
\quad \hline
\quad \mathrm{ran}\ alloc \subseteq freqs \\
\hline
\quad \textsc{Init} \\
\quad alloc = \varnothing \\
\hline
\quad Allocate \\
\quad \ldots \\
\hline
\quad Deallocate \\
\quad \ldots \\
\hline
\end{array}
$$

Initially, no mobiles are allocated frequencies. The definitions of the operations for allocating (*Allocate*) and deallocating (*Deallocate*) frequencies have been elided.

A cell also has a seize unit component which allows a single mobile to seize the cell in order to have a frequency allocated. This is modelled by a class *SeizeUnit* whose state variables are $seize : Boolean$ and $mobile : Mobile$. If *seized*, *mobile* denotes the seizing mobile. (The details of this class will not be elaborated further here.) One particular operation of a cell, modelled by the class *Cell*, is that of paging a mobile. This occurs when there is a call for the mobile. The mobile to be paged must neither be tuned nor seizing.

$$
\begin{array}{|l}
\hline
\textit{Cell} \\
\quad freq\_allocator : FreqAllocator \\
\quad seize\_unit : SeizeUnit \\
\hline
\quad seize\_unit.seized \Rightarrow seize\_unit.mobile \notin \mathrm{dom}\, freq\_allocator.alloc \\
\hline
\quad \textsc{Init} \\
\quad freq\_allocator.\textsc{Init} \wedge seize\_unit.\textsc{Init} \\
\hline
\quad Page \\
\quad mob! : Mobile \\
\quad \hline
\quad mob! \notin \mathrm{dom}\, freq\_allocator.alloc \\
\quad seize\_unit.seized \Rightarrow mob! \neq seize\_unit.mobile \\
\hline
\quad \vdots \\
\hline
\end{array}
$$

The aggregation of the cells in the system is modelled by the class *Cells*. One operation of this class is the *Page* operation which involves all cells simultaneously paging a mobile. This is specifed using the distributed form of the schema conjunction operator.

$$
\begin{array}{|l}
\hline
\textit{Cells} \\
\quad \textit{cells} : \mathbb{F}\,\textit{Cell} \\
\quad \textsc{Init} \\
\quad \forall c : \textit{cells} \bullet c.\textsc{Init} \\
\quad \textit{Page} \mathrel{\widehat=} \bigwedge c : \textit{cells} \bullet c.\textit{Page} \\
\quad \vdots \\
\hline
\end{array}
$$

The system consists of an aggregation of cells and an aggregation of mobiles modelled by a class *Mobiles*. The system operation *Page* involves all cells paging a mobile in the aggregate and the particular mobile being paged (modelled by the *Mobiles* operation *BePaged*). The specification of this operation is given using the parallel operator.

$$
\begin{array}{|l}
\hline
\textit{MobilePhoneSystem} \\
\quad \textit{cells} : \textit{Cells} \\
\quad \textit{mobiles} : \textit{Mobiles} \\
\quad \textsc{Init} \\
\quad \textit{cells}.\textsc{Init} \wedge \textit{mobiles}.\textsc{Init} \\
\quad \textit{Page} \mathrel{\widehat=} \textit{cells}.\textit{Page} \parallel \textit{mobiles}.\textit{BePaged} \\
\quad \vdots \\
\hline
\end{array}
$$

3.1 Verifying properties of the system

The specification of the mobile phone system could include further properties. For example, the sets of mobiles allocated frequencies by individual cells should be disjoint. Rather than include this condition explicitly, however, it is preferable to show that it can be formally derived from the specification.

The proof could proceed by induction, i.e. by showing that the property is true initially and, for each operation, showing that if the property holds in the operation's pre states, it holds in the corresponding post states. Let us consider proving the first part of this, i.e. that the property is true initially.

The first step is to express this formally as a sequent.

$$
\begin{array}{l}
\textit{MobilePhoneSystem} :: \ \textsc{Init} \mid \vdash \forall c_1, c_2 : \textit{cells.cells} \bullet c_1 \neq c_2 \Rightarrow \\
\qquad \mathrm{dom}\ c_1.\textit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\textit{freq\_allocator.alloc} = \varnothing
\end{array}
$$

We then need to prove this sequent using the logic of Section 2.1. As is usual with sequent calculi, proofs are developed backwards. That is, we start with the sequent to be proved and apply to it an inference rule resulting in new sequents (the premisses of the rule) to be proved. This process continues until there are no further sequents to prove (i.e. rules with zero premisses are applied to each remaining sequent).

As an intermediate step it is useful to think of the proof strategy informally. An informal outline of the proof is shown below.

1. The property is equivalent to the following.

$$\begin{array}{l} \mathit{MobilePhoneSystem} :: \textsc{Init};\ c_1, c_2 : \mathit{cells.cells} \mid c_1 \neq c_2 \vdash \\ \qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc} = \varnothing \end{array}$$

The consequent of this property will be true if $\operatorname{dom} c_1.\mathit{freq\_allocator.alloc} = \varnothing$ (since $\varnothing \cap S = \varnothing$).

2. Deduce that *cells* is an object of class *Cells*.

3. From $\textsc{Init}$ deduce that *cells* is in its initial state.

4. From 2 and 3, deduce that $\forall\, c : \mathit{cells.cells} \bullet c.\textsc{Init}$.

5. From 4, deduce that $c_1.\textsc{Init}$.

6. From 2, deduce that *cells.cells* is a set of *Cell* objects.

7. From 6, deduce that c_1 is an object of class *Cell*.

8. From 5 and 7, deduce that $c_1.\mathit{freq\_allocator}.\textsc{Init}$.

9. From 7, deduce that $c_1.\mathit{freq\_allocator}$ is an object of class *FreqAllocator*.

10. From 8 and 9, deduce that $c_1.\mathit{freq\_allocator.alloc} = \varnothing$.

A formal proof based on this strategy and using tactics described in Section 4 is included in Appendix A.

Now let us consider proving that if the property holds in the pre state of *Page* then it holds in the post state of *Page*. Once again, the first step is to express this formally as a sequent.

$$\begin{array}{l} \mathit{MobilePhoneSystem} :: \mathit{Page} \mid \vdash \forall\, c_1, c_2 : \mathit{cells.cells} \bullet c_1 \neq c_2 \Rightarrow \\ \qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc} = \varnothing \\ \qquad \Rightarrow \\ \qquad \forall\, c_1, c_2 : \mathit{cells.cells} \bullet c_1 \neq c_2 \Rightarrow \\ \qquad\qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc}' = \varnothing \end{array}$$

An informal outline of the proof is shown below.

1. The property is equivalent to the following.

$$\begin{array}{l} \mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells.cells} \mid \\ \qquad \forall\, c_1, c_2 : \mathit{cells.cells} \bullet c_1 \neq c_2 \Rightarrow \\ \qquad\qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc} = \varnothing, \\ \qquad\qquad c_1 \neq c_2 \vdash \\ \qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc}' = \varnothing \end{array}$$

2. From the antecedent, deduce that $c_1.\mathit{freq\_allocator.alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc} = \varnothing$.

3. Deduce that *cells* is an object of class *Cells*.

4. From 3, deduce that *cells.cells* is a set of *Cell* objects.

5. From 4, deduce that c_1 and c_2 are objects of class *Cell*.

6. From 5, deduce that $c_1.\mathit{freq\_allocator}$ is an object of class *FreqAllocator*.

7. From 5, deduce that $c_2.\mathit{freq\_allocator}$ is an object of class *FreqAllocator*.

8. From the specification, deduce that $c_1.\mathit{freq\_allocator.alloc}' = c_1.\mathit{freq\_allocator.alloc}$. This involves deducing that $c_1.\mathit{freq\_allocator}$ does not explicitly undergo an operation and

assuming that $c_1.freq\_allocator$ is not referenced by other objects outside the specified system.

9. Similarly, deduce that $c_2.freq\_allocator.alloc' = c_2.freq\_allocator.alloc$.

10. From 2, 8 and 9, deduce the consequent.

Steps 8 and 9 rely on an assumption that objects are not referenced from outside the specified system. This is the basis of the notion of alias control formalised in Section 5. A formal proof based on the above outline is included in Appendix B.

4 Tactics

Tactics are reusable sequences of proof steps used to facilitate and simplify large, complex proofs. They can be used in a proof in the same way as an inference rule. The names of tactics are underlined in the proof presentation to distinguish them from rules. For example, given a tactic *Tac* which performs rules $R_1, \ldots, R_5$, the following proofs are equivalent.

$$
\begin{array}{cl}
S_6 & \\ \hline
S_5 & (\ R_5\ ) \\ \hline
S_4 & (\ R_4\ ) \\ \hline
S_3 & (\ R_3\ ) \\ \hline
S_2 & (\ R_2\ ) \\ \hline
S_1 & (\ R_1\ ) \\ \hline
S_0 & (\ R_0\ )
\end{array}
\qquad\qquad
\begin{array}{cl}
S_5 & \\ \hline
S_1 & (\ \underline{Tac}\ ) \\ \hline
S_0 & (\ R_0\ )
\end{array}
$$

The informal proofs in Section 3 can be used to suggest possible tactics. For example, introducing a property of an initialised object occurs in steps 4, 8 and 10 of the INIT proof. Therefore, a tactic which allows us to introduce the initial condition of an object of arbitrary class would be useful. Also, step 3 of this proof introduces a property of the initial condition of the class within whose context we are reasoning. While this occurs only once in the proof outline, it is general enough to warrant a tactic which performs the necessary steps to achieve it.

Other tactics transform predicates in a sequent to allow more direct application of inference rules. All of the tactics in this section are used either directly or indirectly (i.e. they are called from other tactics) in the formal proofs in the appendices. The rules used in the proofs of the tactics are from the Z Base Standard[1] and [9] or are derived from the stated equivalence by the "semantic-equivalent-lifting" meta-theorem.

4.1 Tactics based on equivalences

The tactics in this section transform a predicate into a more usable form using equivalence properties.

4.1.1 FiniteSet

The *FiniteSet* tactic is used to convert a predicate of the form $x \in \mathbb{F} S$ to a predicate of the form $x \in \mathbb{P} S$. This is necessary so that the *powerset* rule[1] can be applied.

$$\frac{x \in \mathbb{P} S \vdash}{x \in \mathbb{F} S \vdash} \quad (\underline{\mathit{FiniteSet}})$$

Proof:

The given predicate is expressed in terms of a conjunction of the desired predicate and a predicate stating that x is finite. The latter predicate is then removed using the $\wedge\vdash$ and *thin* rules[1].

$$\cfrac{\cfrac{\cfrac{x \in \mathbb{P} S \vdash}{x \in \mathbb{P} S, \exists\, n : \mathbb{N} \bullet \exists\, f : 1 \mathinner{..} n \rightarrow x \bullet \operatorname{ran} f = x \vdash} \; (\mathit{thin})}{x \in \mathbb{P} S \wedge \exists\, n : \mathbb{N} \bullet \exists\, f : 1 \mathinner{..} n \rightarrow x \bullet \operatorname{ran} f = x \vdash} \; (\wedge\vdash)}{x \in \mathbb{F} S \vdash} \; (e)$$

where e **is** $x \in \mathbb{F} S \equiv x \in \mathbb{P} S \wedge \exists\, n : \mathbb{N} \bullet \exists\, f : 1 \mathinner{..} n \rightarrow x \bullet \operatorname{ran} f = x$

4.1.2 SchemaPredicate

The *SchemaPredicate* tactic reduces a schema predicate (denoted by a schema in square brackets) to a predicate.

$$\frac{[d] \wedge p \vdash}{[[d \mid p]] \vdash} \quad (\mathit{SchemaPredicate})$$

Proof:

The square brackets are removed by introducing the substitution of the empty binding into the schema and then using the property $b \odot [S] \equiv b \odot S$. The $[|] \vdash$ rule[1] can then be applied.

$$\cfrac{\cfrac{\cfrac{\cfrac{[d] \wedge p \vdash}{[d \mid p] \vdash} \; ([|] \vdash)}{\langle\!|\,|\!\rangle \odot [d \mid p] \vdash} \; (\langle\!|\,|\!\rangle \odot p \equiv p)}{\langle\!|\,|\!\rangle \odot [[d \mid p]] \vdash} \; (b \odot [S] \equiv b \odot S)}{[[d \mid p]] \vdash} \; (\langle\!|\,|\!\rangle \odot p \equiv p)$$

4.1.3 ObjectSubstitution

The tactic *ObjectSubstitution* evaluates a predicate of the form $a \circledast p$.

Let a be an object of class A with alphabet $x_1 \ldots x_n$ and state variables $x_k \ldots x_n$.

$$\frac{a \in A, p(a.x_1, \ldots, a.x_n, a.x'_k, \ldots, a.x'_n) \vdash}{a \in A, a \circledast p(x_1, \ldots, x_n, x'_k, \ldots, x'_n) \vdash} \quad (\ \underline{ObjectSubstitution}\ )\ [\ q\ ]$$

$$\text{where } q \equiv \alpha(A) = \{x_1, \ldots, x_n\}$$
$$\alpha_1(A) \cup \alpha_2(A) = \{x_k, \ldots, x_n\}$$

Proof:

The predicate is expressed in a form involving binding substitution and evaluated.

$$\begin{array}{ll}
a \in A, p(a.x_1, \ldots, a.x_n, a.x'_k, \ldots, a.x'_n) \vdash & \\
\hline
 & (\ e_2\ ) \\
a \in A, \langle\!|\ x_1 \rightsquigarrow a.x_1, \ldots, x_n \rightsquigarrow a.x_n, x'_k \rightsquigarrow a.x'_k, \ldots, x'_n \rightsquigarrow a.x'_n\ |\!\rangle \odot & \\
\qquad p(x_1, \ldots, x_n, x'_k, \ldots, x'_n) \vdash & \\
\hline
 & (\ \wedge\vdash\ ) \\
a \in A \wedge \langle\!|\ x_1 \rightsquigarrow a.x_1, \ldots, x_n \rightsquigarrow a.x_n, x'_k \rightsquigarrow a.x'_k, \ldots, x'_n \rightsquigarrow a.x'_n\ |\!\rangle \odot & \\
\qquad p(x_1, \ldots, x_n, x'_k, \ldots, x'_n) \vdash & \\
\hline
 & (\ e_1\ )\ [\ q\ ] \\
a \in A \wedge a \circledast p(x_1, \ldots, x_n, x'_k, \ldots, x'_n) \vdash & \\
\hline
 & (\ \wedge\vdash\ ) \\
a \in A, a \circledast p(x_1, \ldots, x_n, x'_k, \ldots, x'_n) \vdash &
\end{array}$$

$$\text{where } q \equiv \alpha(A) = \{x_1, \ldots, x_n\}$$
$$\alpha_1(A) \cup \alpha_2(A) = \{x_k, \ldots, x_n\}$$

e_1 is $a \in A \wedge a \circledast p \equiv$
$a \in A \wedge \langle\!|\ x_1 \rightsquigarrow a.x_1, \ldots, x_n \rightsquigarrow a.x_n, x'_k \rightsquigarrow a.x'_k, \ldots, x'_n \rightsquigarrow a.x'_n\ |\!\rangle \odot p$
e_2 is $\langle\!|\ x_1 \rightsquigarrow y_1, \ldots, x_n \rightsquigarrow y_n\ |\!\rangle \odot p(x_1, \ldots x_n) \equiv p(y_1, \ldots, y_n)$

4.2 Tactics for applying rules

Often a rule requires the predicates to be in a particular form or place in the sequent, i.e. in the antecedent or consequent. The tactics in this section apply a particular rule by first transforming the sequent to the required form.

4.2.1 ApplyLeibniz

The *ApplyLeibniz* tactic replaces occurrences of t in a predicate of the antecedent with s when $s = t$ is also a predicate of the antecedent.

$$\frac{s = t, p(s) \vdash}{s = t, p(t) \vdash} \quad (\ \underline{ApplyLeibniz}\ )$$

It can be applied in both the upward and downward directions, i.e. the following is also a sound application of the tactic.

$$\frac{s = t, p(t) \vdash}{s = t, p(s) \vdash} \quad (\ \underline{ApplyLeibniz}\ )$$

Proof:

For the upward application of the tactic, the predicate is converted into a form involving binding substitution so that the *Leibniz* rule[1] can be applied.

$$\begin{array}{ll} s = t, p(s) \vdash & \\ \hline s = t, \langle\!| x \leadsto s |\!\rangle \odot p(x) \vdash & (\ e\ ) \\ \hline s = t, \langle\!| x \leadsto t |\!\rangle \odot p(x) \vdash & (\ \mathit{Leibniz}\ ) \\ \hline s = t, p(t) \vdash & (\ e\ )\ [\ x \notin \phi(p(t))\ ] \end{array}$$

where e is $\langle\!| x_1 \leadsto y_1, \ldots, x_n \leadsto y_n |\!\rangle \odot p(x_1, \ldots, x_n) \equiv p(y_1, \ldots, y_n)$

For the downward application of the tactic, the proof also requires the use of the equivalence $s = t \equiv t = s$.

$$\begin{array}{ll} s = t, p(t) \vdash & \\ \hline s = t, \langle\!| x \leadsto t |\!\rangle \odot p(x) \vdash & (\ e\ ) \\ \hline t = s, \langle\!| x \leadsto t |\!\rangle \odot p(x) \vdash & (\ s = t \equiv t = s\ ) \\ \hline t = s, \langle\!| x \leadsto s |\!\rangle \odot p(x) \vdash & (\ \mathit{Leibniz}\ ) \\ \hline s = t, \langle\!| x \leadsto s |\!\rangle \odot p(x) \vdash & (\ s = t \equiv t = s\ ) \\ \hline s = t, p(s) \vdash & (\ e\ )\ [\ x \notin \phi(p(s))\ ] \end{array}$$

where e is $\langle\!| x_1 \leadsto y_1, \ldots, x_n \leadsto y_n |\!\rangle \odot p(x_1, \ldots, x_n) \equiv p(y_1, \ldots, y_n)$

4.2.2 ApplyLeibnizThin

Often the predicate $s = t$ is introduced just so the *Leibniz* rule can be applied. The tactic *ApplyLeibnizThin* extends the *ApplyLeibniz* tactic by removing this predicate.

$$\begin{array}{ll} p(s) \vdash & \\ \hline s = t, p(t) \vdash & (\ \underline{\mathit{ApplyLeibnizThin}}\ ) \end{array}$$

Proof:

The predicate is removed using the *thin* rule.

$$\begin{array}{ll} p(s) \vdash & \\ \hline s = t, p(s) \vdash & (\ \mathit{thin}\ ) \\ \hline s = t, p(t) \vdash & (\ \underline{\mathit{ApplyLeibniz}}\ ) \end{array}$$

4.2.3 ApplyForAll

The *ApplyForAll* tactic introduces an instantiation of the predicate of a universal quantification given a declaration of variables of the same type as the quantified variables.

$$\frac{d_y \mid \forall\, d_x \bullet p(x_1, \ldots, x_n), p(y_1, \ldots, y_n) \vdash}{d_y \mid \forall\, d_x \bullet p(x_1, \ldots, x_n) \vdash} \quad (\ \underline{\mathit{ApplyForAll}}\ )$$

where $d_y = d_x[x_1 \leadsto y_1, \ldots, x_n \leadsto y_n]$

Proof:

The declaration is converted into a predicate (using the $d \mid$ rule[1]) which is then converted into a form involving binding membership (using the *schemaexp* rule[1]) so that the $\forall \vdash$ rule[1] can be applied.

$$\begin{array}{ll}
d_y \mid \forall\, d_x \bullet p(x_1, \ldots, y_n), p(y_1, \ldots, y_n) \vdash & \\
\hline
d_y \mid \forall\, d_x \bullet p(x_1, \ldots, x_n), & (\ e\ ) \\
\quad \langle\!| x_1 \leadsto y_1, \ldots, x_n \leadsto y_n |\!\rangle \odot p(x_1, \ldots, x_n) \vdash & \\
\hline
d_y \mid \forall\, d_x \bullet p(x_1, \ldots, x_n), & (\ \mathit{thin}\ ) \\
\quad \langle\!| x_1 \leadsto y_1, \ldots, x_n \leadsto y_n |\!\rangle \in [d_x] & \\
\quad \langle\!| x_1 \leadsto y_1, \ldots, x_n \leadsto y_n |\!\rangle \odot p(x_1, \ldots, x_n) \vdash & \\
\hline
d_y \mid \forall\, d_x \bullet p(x_1, \ldots, x_n), & (\ \forall \vdash\ ) \\
\quad \langle\!| x_1 \leadsto y_1, \ldots, x_n \leadsto y_n |\!\rangle \in [d_x] \vdash & \\
\hline
d_y \mid \forall\, d_x \bullet p(x_1, \ldots, x_n), & (\ \mathit{schemaexp}\ ) \\
\quad \langle\!| x_1 \leadsto y_1, \ldots, x_n \leadsto y_n |\!\rangle \odot [d_x] \vdash & \\
\hline
d_y \mid \forall\, d_x \bullet p(x_1, \ldots, x_n), [d_y] \vdash & (\ e\ ) \\
\hline
d_y \mid \forall\, d_x \bullet p(x_1, \ldots, x_n) \vdash & (\ d \mid)
\end{array}$$

where e is $\langle\!| x_1 \leadsto y_1, \ldots, x_n \leadsto y_n |\!\rangle \odot p(x_1, \ldots, x_n) \equiv p(y_1, \ldots, y_n)$

4.2.4 ApplyForAllThin

Often the predicate $\forall\, d_x \bullet p(x_1, \ldots, x_n)$ is introduced just so the $\forall \vdash$ rule can be applied. The tactic *ApplyForAllThin* extends the *ApplyForAll* tactic by removing this predicate.

$$\frac{d_y \mid p(y_1, \ldots, y_n) \vdash}{d_y \mid \forall\, d_x \bullet p(x_1, \ldots, x_n) \vdash} \quad (\ \underline{\mathit{ApplyForAllThin}}\ )$$

Proof:

The predicate is removed using the *thin* rule.

$$\begin{array}{ll}
d_y \mid p(y_1, \ldots, y_n) \vdash & \\
\hline
d_y \mid \forall\, d_x \bullet p(x_1, \ldots, x_n), p(y_1, \ldots, y_n) \vdash & (\ \mathit{thin}\ ) \\
\hline
d_y \mid \forall\, d_x \bullet p(x_1, \ldots, x_n) \vdash & (\ \underline{\mathit{ApplyForAll}}\ )
\end{array}$$

4.2.5 ApplyImplies

The *ApplyImplies* tactic introduces the consequent of an implication given the antecedent.

$$\frac{p, q \vdash}{p, p \Rightarrow q \vdash} \quad (\ \underline{ApplyImplies}\ )$$

Proof:

The consequent is introduced using the $\Rightarrow\vdash$ rule[1].

$$\frac{\begin{array}{l} p, p \Rightarrow q \vdash p \\ p, q \vdash \end{array}}{p, p \Rightarrow q \vdash} \quad (\ \Rightarrow\vdash\ )$$

The second premiss is of the desired form. The first premiss is discharged using the *assumption* rule[1].

$$\frac{}{p, p \Rightarrow q \vdash p} \quad (\ assumption\ )$$

4.2.6 ApplyImpliesThin

The tactic *ApplyImpliesThin* extends the *ApplyImplies* tactic by removing the antecedent of the implication.

$$\frac{q \vdash}{p, p \Rightarrow q \vdash} \quad (\ \underline{ApplyImpliesThin}\ )$$

Proof:

The antecedent of the implication is removed using the *thin* rule.

$$\frac{\dfrac{q \vdash}{p, q \vdash}\ (\ thin\ )}{p, p \Rightarrow q \vdash} \quad (\ \underline{ApplyImplies}\ )$$

4.2.7 ApplyToConsequent

The tactic *ApplyToConsequent* applies a rule or tactic which operates on a predicate in the antecedent, to a predicate in the consequent. The rule or tactic must be able to be applied in both the upward and downward directions. The name of the rule or tactic is supplied as a parameter.

Let

$$\frac{q \vdash}{p \vdash} \quad (\ Name\ )$$

be a rule or tactic which can be applied in the upward and downward directions.

$$\frac{\vdash q}{\vdash p} \quad (\underline{ApplyToConsequent}(Name))$$

Proof:

The predicate q is introduced using the *cut* rule[1].

$$\frac{\begin{array}{c} q \vdash p \\ \vdash p, q \end{array}}{\vdash p} \quad (cut)$$

The first premiss is discharged by downward application of *Name* followed by the *assumption* rule.

$$\frac{\overline{\quad p \vdash p \quad} \; (assumption)}{q \vdash p} \quad (Name)$$

The second premiss is converted to the desired form by the *thin* rule.

$$\frac{\vdash q}{\vdash p, q} \quad (thin)$$

4.2.8 ApplyToAntecedent

The tactic *ApplyToAntecedent* is similar to *ApplyToConsequent*. It applies a rule or tactic which operates on a predicate in the consequent, to a predicate in the antecedent. The rule or tactic must be able to be applied in both the upward and downward directions. The name of the rule or tactic is supplied as a parameter.

Let

$$\frac{\vdash q}{\vdash p} \quad (Name)$$

be a rule or tactic which can be applied in the upward and downward directions.

$$\frac{q \vdash}{p \vdash} \quad (\underline{ApplyToAntecedent}(Name))$$

Proof:

The predicate q is introduced using the *cut* rule.

$$\frac{\begin{array}{c} p, q \vdash \\ p \vdash q \end{array}}{p \vdash} \quad (cut)$$

The second premiss is discharged by downward application of *Name* followed by the *assumption* rule.

$$\frac{\overline{p \vdash p} \; (\ assumption\ )}{p \vdash q} \; (\ Name\ )$$

The first premiss is converted to the desired form by the *thin* rule.

$$\frac{q \vdash}{p, q \vdash} \; (\ thin\ )$$

4.3 Tactics for introducing class properties

The tactics in this section introduce properties of the class in which reasoning is taking place. For example, we can introduce the definition of the class' schemas or, if we are reasoning in the context of a particular class schema, we can introduce the schema's definition, or part thereof, as a predicate.

4.3.1 IntroduceStateDef

The tactic *IntroduceStateDef* introduces the fully-expanded state schema definition of a class. That is, it introduces a predicate equating $\mathit{S\scriptstyle TATE}$ (a keyword introduced in the logic to identify the state schema of a class) to an expression which does not refer to other schemas of the class.

Let class A have state schema $[d \mid p]$ and inherit classes $B_1, \ldots, B_n$ with fully-expanded state schemas $S_1, \ldots, S_n$ respectively. This latter condition is captured by the proviso. (ι is a meta-function which returns the set of inherited classes of a class. The predicates of the form $\phi(S_i) \subseteq \alpha(A_i)$ ensure that the schemas S_i are expressed in fully-expanded form, i.e. in terms of $\alpha(A_i)$ only and not in terms of references to the state, initial state or operations of A_i).

$$\frac{A :: \mathit{S\scriptstyle TATE} = [S_1 \wedge \ldots \wedge S_n;\ d \mid p] \vdash}{A :: \vdash} \quad (\ \underline{\mathit{IntroduceStateDef}}\ )\ [\ q\ ]$$

$$\text{where } q \equiv \begin{array}{l} \iota(A) = \{B_1, \ldots, B_n\} \\ \phi(S_1) \subseteq \alpha(B_1) \\ \ldots \\ \phi(S_n) \subseteq \alpha(B_n) \end{array}$$

Proof:

After application of *StateDef* [9][9], we introduce a predicate equating the schema $\uparrow \mathit{S\scriptstyle TATE}$ (denoting the inherited part of the state schema) with the conjunction of the fully-expanded state schemas of the inherited classes.

[9]The rules used in this section are identical to those in [9] except that generic parameters have been ignored.

$$\dfrac{\dfrac{\begin{array}{l} A :: \textit{State} = [\uparrow\textit{State};\ d \mid p], \uparrow\textit{State} = S_1 \wedge \ldots \wedge S_n \vdash \\ A :: \textit{State} = [\uparrow\textit{State};\ d \mid p] \vdash \uparrow\textit{State} = S_1 \wedge \ldots \wedge S_n \end{array}}{A :: \textit{State} = [\uparrow\textit{State};\ d \mid p] \vdash} \ (\ \textit{cut}\ )}{A :: \vdash} \ (\ \textit{StateDef}\ )$$

The first premiss is converted to the desired form by application of the tactic *ApplyLeibnizThin*.

$$\dfrac{A :: \textit{State} = [S_1 \wedge \ldots \wedge S_n;\ d \mid p] \vdash}{A :: \textit{State} = [\uparrow\textit{State};\ d \mid p], \uparrow\textit{State} = S_1 \wedge \ldots \wedge S_n \vdash} \ (\ \underline{\textit{ApplyLeibnizThin}}\ )$$

To discharge the second premiss we need to show that the introduced predicate is true. To do this, we apply the $\uparrow$*State* rule[9]. The proviso of this rule becomes the proviso of the tactic.

$$\dfrac{B_1 :: \vdash \textit{State} = S_1 \quad \ldots \quad B_n :: \vdash \textit{State} = S_n}{A :: \textit{State} = [\uparrow\textit{State};\ d \mid p] \vdash \uparrow\textit{State} = S_1 \wedge \ldots \wedge S_n} \ (\ \uparrow\textit{State}\ )\ [\ q\ ]$$

$$\begin{array}{ll} \text{where } q \equiv & \iota(A) = \{B_1, \ldots, B_n\} \\ & \phi(S_1) \subseteq \alpha(B_1) \\ & \ldots \\ & \phi(S_n) \subseteq \alpha(B_n) \end{array}$$

Then the tactic *IntroduceStateDef* is applied to each of the resulting premisses. The recursion to *IntroduceStateDef* will end when we reach classes with no inherited classes. (For simplicity of presentation, the provisos of these tactic applications have been elided.)

$$\dfrac{}{B_1 :: \vdash \textit{State} = S_1} \ (\ \underline{\textit{IntroduceStateDef}}\ )\ [\ \ldots\ ]$$

...

$$\dfrac{}{B_n :: \vdash \textit{State} = S_n} \ (\ \underline{\textit{IntroduceStateDef}}\ )\ [\ \ldots\ ]$$

4.3.2 IntroduceInitDef

The tactic *IntroduceInitDef* is similar to *IntroduceStateDef*. It introduces the fully-expanded initial state schema definition of a class.

Let class A have initial state schema $[p]$, fully-expanded state schema S and inherit classes $B_1, \ldots, B_n$ with fully-expanded initial state schemas $S_1, \ldots, S_n$ respectively. The latter condition is captured by the proviso.

$$\dfrac{A :: \textit{Init} = S_1 \wedge \ldots \wedge S_n \wedge [S \mid p] \vdash}{A :: \vdash} \ (\ \underline{\textit{IntroduceInitDef}}\ )\ [\ q\ ]$$

where $q \equiv \iota(A) = \{B_1, \ldots, B_n\}$
$\phi(S_1) \subseteq \alpha(B_1)$
$\ldots$
$\phi(S_n) \subseteq \alpha(B_n)$

Proof:

After application of *InitDef* [9], we introduce a predicate equating the schema $\uparrow$INIT (denoting the inherited part of the initial state schema) with the conjunction of the fully-expanded initial state schemas of the inherited classes.

$$\cfrac{\cfrac{A :: \textit{Init} = \uparrow\textit{Init} \wedge [\textit{State} \mid p], \uparrow\textit{Init} = S_1 \wedge \ldots \wedge S_n \vdash \quad A :: \textit{Init} = \uparrow\textit{Init} \wedge [\textit{State} \mid p] \vdash \uparrow\textit{Init} = S_1 \wedge \ldots \wedge S_n}{A :: \textit{Init} = \uparrow\textit{Init} \wedge [\textit{State} \mid p] \vdash} \ (\ \textit{cut}\ )}{A :: \vdash} \ (\ \textit{InitDef}\ )$$

The first premiss is converted to the desired form by application of the tactic *ApplyLeibnizThin* followed by *IntroduceStateDef* and another application of *ApplyLeibnizThin*.

$$\cfrac{\cfrac{\cfrac{A :: \textit{Init} = S_1 \wedge \ldots \wedge S_n \wedge [S \mid p]}{A :: \textit{Init} = S_1 \wedge \ldots \wedge S_n \wedge [\textit{State} \mid p], \textit{State} = S \vdash} \ (\ \underline{\textit{ApplyLeibnizThin}}\ )}{A :: \textit{Init} = S_1 \wedge \ldots \wedge S_n \wedge [\textit{State} \mid p] \vdash} \ (\ \underline{\textit{IntroduceStateDef}}\ )\ [\ldots]}{A :: \textit{Init} = \uparrow\textit{Init} \wedge [\textit{State} \mid p], \uparrow\textit{Init} = S_1 \wedge \ldots \wedge S_n \vdash} \ (\ \underline{\textit{ApplyLeibnizThin}}\ )$$

To discharge the second premiss we need to show that the introduced predicate is true. To do this, we apply the $\uparrow$INIT rule[9]. The proviso of this rule becomes the proviso of the tactic.

$$\cfrac{B_1 :: \vdash \textit{Init} = S_1 \quad \ldots \quad B_n :: \vdash \textit{Init} = S_n}{A :: \textit{Init} = \uparrow\textit{Init} \wedge [\textit{State} \mid p] \vdash \uparrow\textit{Init} = S_1 \wedge \ldots \wedge S_n} \ (\ \uparrow\textit{Init}\ )\ [\ q\ ]$$

where $q \equiv \iota(A) = \{B_1, \ldots, B_n\}$
$\phi(S_1) \subseteq \alpha(A_1)$
$\ldots$
$\phi(S_n) \subseteq \alpha(B_n)$

Then the tactic *IntroduceInitDef* is applied to each of the resulting premisses. (As with *IntroduceStateDef*, the recursion to *IntroduceInitDef* will end when we reach classes with no inherited classes.)

$$\cfrac{}{B_1 :: \vdash \textit{Init} = S_1} \ (\ \underline{\textit{IntroduceInitDef}}\ )\ [\ldots]$$

$\ldots$

$$\cfrac{}{B_n :: \vdash \textit{Init} = S_n} \ (\ \underline{\textit{IntroduceInitDef}}\ )\ [\ldots]$$

4.3.3 IntroduceInit

The tactic *IntroduceInit* introduces the initial state schema as a predicate when it occurs as a declaration.

Let class A have initial state schema $[p]$.

$$\frac{\textsc{Init} \mid [\uparrow\textsc{Init} \wedge [\textsc{State} \mid p]] \vdash}{\textsc{Init} \mid \vdash} \quad (\ \underline{\mathit{IntroduceInit}}\ )$$

Proof:

The predicate form of the declaration is replaced by the definition of the initial state schema obtained by applying the *InitDef* rule. Alternatively, we could have used the definition of the initial state schema obtained by applying the *IntroduceInitDef* tactic, i.e. the fully-expanded definition. However, the simpler form is preferred for defining the tactic in Section 4.3.4.

$$\begin{array}{ll}
\textsc{Init} \mid [\uparrow\textsc{Init} \wedge [\textsc{State} \mid p]] \vdash & \\ \hline
\textsc{Init} \mid [\textsc{Init}], \textsc{Init} = \uparrow\textsc{Init} \wedge [\textsc{State} \mid p] \vdash & (\ \underline{\mathit{ApplyLeibnizThin}}\ ) \\ \hline
\textsc{Init} \mid [\textsc{Init}] \vdash & (\ \mathit{InitDef}\ ) \\ \hline
\textsc{Init} \mid \vdash & (\ d \mid\ )
\end{array}$$

4.3.4 IntroduceInitPredicate

Often we would like to introduce just the explicit initial condition. The tactic *IntroduceInit-Predicate* extends *IntroduceInit* to do this.

Let class A have initial state schema $[p]$.

$$\frac{\textsc{Init} \mid p \vdash}{\textsc{Init} \mid \vdash} \quad (\ \underline{\mathit{IntroduceInitPredicate}}\ )$$

Proof:

The constituent parts of the predicate representing the initial state schema are separated (using the $[\bigwedge] \vdash$ and $\wedge\vdash$ rules[1] together with the *SchemaPredicate* tactic) and those other than the explicit initial condition are removed.

$$\begin{array}{ll}
\textsc{Init} \mid p \vdash & \\ \hline
\textsc{Init} \mid [\uparrow\textsc{Init}], [\textsc{State}], p \vdash & (\ \mathit{thin}\ ) \\ \hline
\textsc{Init} \mid [\uparrow\textsc{Init}], [\textsc{State}] \wedge p \vdash & (\ \wedge\vdash\ ) \\ \hline
\textsc{Init} \mid [\uparrow\textsc{Init}], [[\textsc{State} \mid p]] \vdash & (\ \underline{\mathit{SchemaPredicate}}\ ) \\ \hline
\textsc{Init} \mid [\uparrow\textsc{Init} \wedge [\textsc{State} \mid p]] \vdash & (\ [\bigwedge] \vdash\ ) \\ \hline
\textsc{Init} \mid \vdash & (\ \underline{\mathit{IntroduceInit}}\ )
\end{array}$$

4.4 Tactics for introducing properties of objects

The tactics in this section introduce properties of objects referenced from the class in which we are reasoning or expand expressions involving such objects.

4.4.1 IntroduceObjectProperty

The tactic *IntroduceObjectProperty* introduces the properties of an object derived from the local environment and state schema of its class.

Let class A have local environment represented by the predicate p and fully-expanded state schema S.

$$\frac{a \in A, a \circledast (p \wedge [\Delta S]) \vdash}{a \in A \vdash} \quad (\ \underline{\mathit{IntroduceObjectProperty}}\ )\ [\ \phi(S) \subseteq \alpha(A)\ ]$$

We don't evaluate the introduced predicate using *ObjectSubstitution* so that the tactic can be extended as in Section 4.4.2.

Proof:

The properties of the object are introduced using the *cut* rule.

$$\frac{\begin{array}{l} a \in A, a \circledast (p \wedge [\Delta S]) \vdash \\ a \in A \vdash a \circledast (p \wedge [\Delta S]) \end{array}}{a \in A \vdash} \quad (\ \mathit{cut}\ )$$

The first premiss is of the desired form. To discharge the second premiss, we apply the *ClassMembership* rule[9]. The proviso of this rule becomes the proviso of the tactic.

$$\frac{\begin{array}{l} a \in A \vdash a \circledast (p \wedge [\Delta S]), a \in A \\ A :: \vdash \textsc{State} = S \end{array}}{a \in A \vdash a \circledast (p \wedge [\Delta S])} \quad (\ \mathit{ClassMembership}\ )\ [\ \phi(S) \subseteq \alpha(A)\ ]$$

The first premiss is discharged using the *assumption* rule.

$$\frac{}{a \in A \vdash a \circledast (p \wedge [\Delta S]), a \in A} \quad (\ \mathit{assumption}\ )$$

The second premiss is discharged by applying the tactic *IntroduceStateDef* followed by the *assumption* rule.

$$\frac{\dfrac{}{A :: \textsc{State} = S \vdash \textsc{State} = S}\ (\ \mathit{assumption}\ )}{A :: \vdash \textsc{State} = S} \quad (\ \underline{\mathit{IntroduceStateDef}}\ )\ [\ \ldots\ ]$$

4.4.2 IntroduceObjectLocalProperty

Often we would like to introduce just the properties of an object derived from the local environment of its class. The tactic *IntroduceObjectLocalProperty* is an extension of *IntroduceObjectProperty* to do this. Similarly, a tactic to introduce just the properties of an object derived from the state schema of its class could also be defined.

Let class A have local environment represented by the predicate p.

$$\frac{a \in A, q \vdash}{a \in A \vdash} \quad (\ \underline{\mathit{IntroduceObjectLocalProperty}}\ )$$

where $q \equiv a \circledast p$

Proof:
The properties derived from the local environment and the state are separated and the latter are removed.

$$
\begin{array}{ll}
a \in A, q \vdash & \\ \hline
a \in A, a \circledast p \vdash & (\ \underline{\mathit{ObjectSubstitution}}\ )\ [\ \ldots\ ] \\ \hline
a \in A, a \circledast p, a \circledast [\Delta S] \vdash & (\ \mathit{thin}\ ) \\ \hline
a \in A, a \circledast p \wedge a \circledast [\Delta S] \vdash & (\ \wedge\vdash\ ) \\ \hline
a \in A, a \circledast (p \wedge [\Delta S]) \vdash & (\ a \circledast (p \wedge q) \equiv a \circledast p \wedge a \circledast q\ ) \\ \hline
a \in A \vdash & (\ \underline{\mathit{IntroduceObjectProperty}}\ )\ [\ \ldots\ ]
\end{array}
$$

4.4.3 ExpandObjectInit

The tactic *ExpandObjectInit* expands a predicate of the form $a.\textsc{Init}$ where a is an object. Let class A have fully-expanded initial state schema S.

$$
\frac{a \in A, a \circledast [S] \vdash}{a \in A, a.\textsc{Init} \vdash} \quad (\ \underline{\mathit{ExpandObjectInit}}\ )\ [\ \phi(S) \subseteq \alpha(A)\ ]
$$

As with *IntroduceObjectProperty*, we don't evaluate the resulting predicate using *ObjectSubstitution* so that the tactic can be extended as in Section 4.4.4.

Proof:
A predicate stating that $a.\textsc{Init}$ is equivalent to $a \circledast S$ is introduced.

$$
\frac{\begin{array}{l} a \in A, a.\textsc{Init}, a.\textsc{Init} \Leftrightarrow a \circledast [S] \\ a \in A, a.\textsc{Init} \vdash a.\textsc{Init} \Leftrightarrow a \circledast [S] \end{array}}{a \in A, a.\textsc{Init} \vdash} \quad (\ \mathit{cut}\ )
$$

The second premiss, i.e. the introduced predicate, is discharged by applying the *Initialisation* rule[9] followed by application of the *assumption* rule and the *IntroduceInitDef* tactic to the resulting predicates. The proviso of this rule becomes the proviso of the tactic.

$$
\frac{\begin{array}{l} a \in A, a.\textsc{Init} \vdash a.\textsc{Init} \Leftrightarrow a \circledast [S], a \in A \\ A :: \vdash \textsc{Init} = S \end{array}}{a \in A, a.\textsc{Init} \vdash a.\textsc{Init} \Leftrightarrow a \circledast [S]} \quad (\ \mathit{Initialisation}\ )\ [\ \phi(S) \subseteq \alpha(A)\ ]
$$

$$
\frac{}{a \in A, a.\textsc{Init} \vdash a.\textsc{Init} \Leftrightarrow a \circledast [S], a \in A} \quad (\ \mathit{assumption}\ )
$$

$$
\begin{array}{ll}
 & \\ \hline
A :: \textsc{Init} = S \vdash \textsc{Init} = S & (\ \mathit{assumption}\ ) \\ \hline
A :: \vdash \textsc{Init} = S & (\ \underline{\mathit{IntroduceInitDef}}\ )\ [\ \ldots\ ]
\end{array}
$$

The first premiss is converted to the desired form using the $\Leftrightarrow\vdash$ rule and the *ApplyImplies-Thin* tactic.

$$\begin{array}{ll}
a \in A,\, a \circledast [S] \vdash & \\
\hline
a \in A,\, a.\textsc{Init},\, a.\textsc{Init} \Rightarrow a \circledast [S] \vdash & (\ \underline{ApplyImpliesThin}\ ) \\
\hline
a \in A,\, a.\textsc{Init},\, a.\textsc{Init} \Rightarrow a \circledast [S],\, a \circledast [S] \Rightarrow a.\textsc{Init} \vdash & (\ thin\ ) \\
\hline
a \in A,\, a.\textsc{Init},\, a.\textsc{Init} \Leftrightarrow a \circledast [S] \vdash & (\ \Leftrightarrow\vdash\ )
\end{array}$$

4.4.4 ExpandObjectInitPredicate

Often we would like to introduce just the initial condition stated explicitly in an object's class when we have a predicate $a.\textsc{Init}$. The tactic *ExpandObjectInitPredicate* extends *ExpandObjectInit* to do this.

Let class A have initial state schema $[p]$.

$$\frac{a \in A,\, q \vdash}{a \in A,\, a.\textsc{Init} \vdash} \quad (\ \underline{ExpandObjectInitPredicate}\ )$$

where $q \equiv a \circledast p$

Proof:

The initial state schema is separated into its constituent properties and those derived from the state schema and inherited initial state schemas are removed.

$$\begin{array}{ll}
a \in A,\, q \vdash & \\
\hline
a \in A,\, a \circledast p \vdash & (\ \underline{ObjectSubstitution}\ )\ [\ \ldots\ ] \\
\hline
a \in A,\, a \circledast S_1,\, a \circledast [S],\, a \circledast p \vdash & (\ thin\ ) \\
\hline
a \in A,\, a \circledast S_1,\, a \circledast [S] \wedge a \circledast p \vdash & (\ \wedge\vdash\ ) \\
\hline
a \in A,\, a \circledast [S_1],\, a \circledast [S \mid p] \vdash & (\ a \circledast [d \mid p] \equiv a \circledast [d] \wedge a \circledast p\ ) \\
\hline
a \in A,\, a \circledast [S_1],\, a \circledast [[S \mid p]] \vdash & (\ a \circledast [S] \equiv a \circledast S\ ) \\
\hline
a \in A,\, a \circledast [S_1] \wedge a \circledast [[S \mid p]] \vdash & (\ \wedge\vdash\ ) \\
\hline
a \in A,\, a \circledast [S_1 \bigwedge [S \mid p]] \vdash & (\ a \circledast [S \bigwedge T] \equiv a \circledast [S] \wedge a \circledast [T]\ ) \\
\hline
a \in A,\, a.\textsc{Init} \vdash & (\ \underline{ExpandObjectInit}\ )\ [\ \ldots\ ]
\end{array}$$

5 Alias control

Aliasing occurs as a result of two or more object references referring to the same object. These references can occur either in the same class or in different classes which make up the specification. Aliasing restricts reasoning by reducing what we can deduce about a particular class in isolation. For example, consider the following where C is a class with operation Op.

A

$$x, y : C$$
$$x \neq y$$

$$OpA \mathrel{\widehat=} x.Op$$

B

$$a : A$$
$$z : C$$

$$OpB \mathrel{\widehat=} z.Op \land a.OpA$$

If, in class B, $a.y$ and z refer to the same object then, when OpB is performed, effectively both $a.x$ and $a.y$ undergo operation Op. Hence, when reasoning about the operation OpA in class A, we cannot deduce that the object referenced by y remains unchanged.

In order to practically reason about a system specification, it is necessary to limit aliasing. One way to do this is to assume that no aliasing occurs between objects which are referenced by the specified system and objects referenced by other systems with which the specified system may be composed. That is, objects which are not changed explicitly by operations in the system class remain unchanged when those operations occur.

To do this we introduce a meta-object Sys whose class is the system class. Formally, this class could be defined as the final class of a specification.

Next we introduce a meta-function χ_{OP} which returns the set of pairs of objects changed by an operation and their classes. If the operation involves the choice operator $[]$ then this set contains all object/class pairs of objects changed by any of the alternative operations.

$$
\begin{array}{l}
\chi_{OP}\left[\Delta(x_1, \ldots, x_n)\ d \mid p\right] = \varnothing \\
\chi_{OP}\left[d \mid p\right] = \varnothing \\
\chi_{OP}(a.Op) = \{(a, A)\} \cup a \circledast \chi(A, Op) \quad \text{where } a \in A \\
\chi_{OP}(OP_1 \land OP_2) = \chi_{OP}(OP_1) \cup \chi_{OP}(OP_2) \\
\chi_{OP}(OP_1 \parallel OP_2) = \chi_{OP}(OP_1) \cup \chi_{OP}(OP_2) \\
\chi_{OP}(OP_1 \; [] \; OP_2) = \chi_{OP}(OP_1) \cup \chi_{OP}(OP_2) \\
\chi_{OP}(OP_1 \bullet OP_2) = \chi_{OP}(OP_1) \cup \chi_{OP}(OP_2) \\
\chi_{OP}(OP[\lambda]) = \langle\!|\ \lambda\ |\!\rangle \odot \chi_{OP}(OP) \\
\chi_{OP}(OP \setminus (x_1, \ldots, x_n)) = \chi_{OP}(OP) \\
\chi_{OP}(\bigwedge d \mid p \bullet OP) = \chi_{OP}(OP) \\
\chi_{OP}(\big\| \ d \mid p \bullet OP) = \chi_{OP}(OP) \\
\chi_{OP}(\big[\big] \ d \mid p \bullet OP) = \chi_{OP}(OP)
\end{array}
$$

The meta-function χ returns the objects changed by an operation on the system object of a specification. It is defined in terms of the meta-function χ_d which returns the set of objects changed by the explicitly declared operation.

Given a class A with operation definition $Op \mathrel{\widehat=} OP$,

$$\chi_d(A, Op) = \chi_{OP}(OP)$$

To account for the objects changed by the inherited parts of operations we have the following. (Ω_d is a meta-function which returns the set of operations explicitly defined in a class. Ω is a meta-function which returns the total set of operations of a class, i.e. both those defined explicitly and those defined implicitly via inheritance.)

$$
\begin{aligned}
\chi(A[\lambda], Op) &= \langle\!|\ \lambda\ |\!\rangle \odot \chi(A, Op)\\
\chi(B, Op) &= \chi_d(B, Op) \cup \chi(A_1, Op) \cup \ldots \cup \chi(A_k, Op) \quad \text{if } Op \in \Omega_d(B)\\
&= \chi(A_1, Op) \cup \ldots \cup \chi(A_k, Op) \quad \text{if } Op \notin \Omega_d(B)\\
&\text{where } \iota(B) = \{A_1, \ldots, A_n\}\\
&\qquad Op \in \Omega(A_1) \cap \ldots \cap \Omega(A_k)\\
&\qquad Op \notin \Omega(A_{k+1}) \cup \ldots \cup \Omega(A_n)
\end{aligned}
$$

The following inference rule captures the notion of alias control. When an object is not changed by an operation of the system object then we can introduce the predicate that states that its primary state variables are unchanged. The object must not be aliased to any other object, of the same or a compatible class, changed by the operation. This rule is used in the formal proof in Appendix B.

$$
\frac{\begin{array}{l}
B :: a_1 : A_1;\ Op \mid \vdash a \neq a_1\\
\ldots\\
B :: a_m : A_m;\ Op \mid \vdash a \neq a_m\\
B :: Op \mid a \in A,\ a.x_1' = a.x_1 \wedge \ldots \wedge a.x_n' = a.x_n \vdash
\end{array}}{B :: Op \mid a \in A \vdash} \quad (\ \mathit{UnchangedObject}\ )\ [\ p\ ]
$$

$$
\begin{aligned}
\text{where } p &\equiv \mathit{Sys} \in B\\
&\iota^*(A) = \{B_1, \ldots, B_k\}\\
&\chi(B, Op) \rhd \{A, \downarrow B_1, \ldots, \downarrow B_k\} = \{(a_1, A_1), \ldots, (a_m, A_m)\}\\
&\alpha_1(A) = \{x_1, \ldots, x_n\}
\end{aligned}
$$

The third predicate of the proviso states that the set of objects referenced by $a_1, \ldots, a_m$ are all those of the same or a compatible class as a which are changed (or possibly changed when choice is involved) by Op. The declarations $a_i : A_i$ are necessary in the premisses in the case when a_i has been hidden in Op or is only declared in a distributed operator expression. Methods for utilising the predicates associated with such hidden or locally declared object references when proving these premisses need to be investigated.

6 Conclusions

This paper has extended the logic for Object-Z in [9] with tactics and a notion of alias control to facilitate and simplify the formal verification of system properties. The extensions are illustrated through a case study based on the mobile phone system specification in [8].

The extensions in this paper are only indicative of the type of extensions required for a fully practical proof system for Object-Z. The tactics presented were only those required by the proofs in the appendices. A much more extensive library of tactics needs to be developed. Furthermore, the tactics presented only combine inference rules sequentially or, in the case of multiple premisses to a rule, in parallel. More complex tactics could be derived by, for example, allowing choice between alternative rules.

The notion of alias control prevented aliasing between objects in the specified system and other objects in the system of which the specified system is a component. Aliasing can, however, occur between objects in the specified system. In small specifications, we could reason about such aliasing using an exhaustive case analysis covering all the possible combinations of equalities and inequalities between object references in the specification. This would, however, be impractical in even medium-sized specifications due to the large number of object references which could exist. Furthermore, aliasing inhibits modular reasoning. Hence, a further means of limiting and controlling aliasing is seen as desirable.

Acknowledgements

The author would like to thank Roger Duke, Wendy Johnston and Gordon Rose for many helpful discussions. This work is supported by the Australian Research Council (ARC).

References

[1] S.M. Brien and J.E. Nicholls. Z Base Standard: Version 1. Technical Report PRG-107, Oxford University Computing Laboratory, 1992.

[2] J. Dong and R. Duke. Class union and polymorphism. In C. Mingins, W. Haebich, J. Potter, and B. Meyer, editors, *Technology of Object-Oriented Languages and Systems (TOOLS 12 & 9)*, pages 181–190. Prentice-Hall International, 1993.

[3] J. Dong and R. Duke. The geometry of object containment. *Object-Oriented Systems (OOS)*, 2:41–63, 1995.

[4] R. Duke, G. Rose, and G. Smith. Object-Z: A specification language advocated for the description of standards. *Computer Standards and Interfaces*, 17:511–533, 1995.

[5] A. Griffiths. An extended semantic foundation for Object-Z. Technical Report 95-39, Software Verification Research Centre, Department of Computer Science, University of Queensland, Australia, 1995.

[6] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and Z.* Series in Computer Science. Prentice-Hall International, 1990.

[7] G. Rose. Object-Z. In S. Stepney, R. Barden, and D. Cooper, editors, *Object Orientation in Z*, Workshops in Computing, pages 59–77. Springer-Verlag, 1992.

[8] G. Rose and R. Duke. An Object-Z specification of a mobile phone system. In K. Lano and H. Haughton, editors, *Object-Oriented Specification Case Studies*, pages 110–129. Prentice-Hall International, 1993.

[9] G. Smith. Extending $\mathcal{W}$ for Object-Z. In J. Bowen and M. Hinchey, editors, *9th International Conference of Z Users*, volume 967 of *Lecture Notes in Computer Science*, pages 276–295. Springer-Verlag, 1995.

[10] G. Smith. A logic for Object-Z (additional rules). Technical Report 95-26, Software Verification Research Centre, Department of Computer Science, University of Queensland, Australia, 1995.

[11] J.M. Spivey. *The Z Notation: A Reference Manual (2nd Ed.).* Series in Computer Science. Prentice-Hall International, 1992.

Appendix A

The following formal proof follows the informal proof outline for $\textsc{Init}$ given in Section 3 and uses tactics defined in Section 4. References to proof steps in the informal proof are given in parentheses. For example, (1) refers to proof step 1.

We begin by simplifying the consequent. This is done using the rules for universal quantification and implication elimination. (1)

$$
\begin{array}{ll}
\mathit{MobilePhoneSystem} :: \ \textsc{Init};\ c_1, c_2 : \mathit{cells.cells} \mid \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\hline
 & (\ \mathit{thin}\ ) \\
\mathit{MobilePhoneSystem} :: \ \textsc{Init};\ c_1, c_2 : \mathit{cells.cells} \mid c_1 \neq c_2 \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\hline
 & (\ \vdash\Rightarrow\ ) \\
\mathit{MobilePhoneSystem} :: \ \textsc{Init};\ c_1, c_2 : \mathit{cells.cells} \vdash c_1 \neq c_2 \Rightarrow & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\hline
 & (\ \vdash \forall\ ) \\
\mathit{MobilePhoneSystem} :: \ \textsc{Init} \mid \vdash \forall\, c_1, c_2 : \mathit{cells.cells} \bullet c_1 \neq c_2 \Rightarrow & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing &
\end{array}
$$

We then introduce the property that *cells* is an object of class *Cells*. (2)

$$
\begin{array}{ll}
\mathit{MobilePhoneSystem} :: \textsc{Init};\ c_1, c_2 : \mathit{cells.cells} \mid \mathit{cells} \in \mathit{Cells} \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\hline
 & (\ [n : s] \vdash\ ) \\
\mathit{MobilePhoneSystem} :: \textsc{Init};\ c_1, c_2 : \mathit{cells.cells} \mid [\mathit{cells} : \mathit{Cells}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\hline
 & (\ \mathit{thin}\ ) \\
\mathit{MobilePhoneSystem} :: \textsc{Init};\ c_1, c_2 : \mathit{cells.cells} \mid [\mathit{cells} : \mathit{Cells}], & \\
\quad [\mathit{mobiles} : \mathit{Mobiles}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\hline
 & (\ \wedge\vdash\ ) \\
\mathit{MobilePhoneSystem} :: \textsc{Init};\ c_1, c_2 : \mathit{cells.cells} \mid & \\
\quad [\mathit{cells} : \mathit{Cells}] \wedge [\mathit{mobiles} : \mathit{Mobiles}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\hline
 & (\ [D;\ D] \vdash\ ) \\
\mathit{MobilePhoneSystem} :: \textsc{Init};\ c_1, c_2 : \mathit{cells.cells} \mid & \\
\quad [\mathit{cells} : \mathit{Cells};\ \mathit{mobiles} : \mathit{Mobiles}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\hline
 & (\ \mathit{LocalDefs}\ ) \\
\mathit{MobilePhoneSystem} :: \textsc{Init} \mid c_1, c_2 : \mathit{cells.cells} \mid \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing &
\end{array}
$$

From $\text{I}\textsc{nit}$ we introduce the property that *cells* is in its initial state. (3)

$$\cfrac{\cfrac{\cfrac{\begin{array}{l} \mathit{MobilePhoneSystem} :: c_1, c_2 : \mathit{cells}.\mathit{cells} \mid \mathit{cells} \in \mathit{Cells}, \mathit{cells}.\textit{I}\textsc{nit} \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \end{array}}{\begin{array}{l} \mathit{MobilePhoneSystem} :: \textit{I}\textsc{nit};\ c_1, c_2 : \mathit{cells}.\mathit{cells} \mid \mathit{cells} \in \mathit{Cells}, \mathit{cells}.\textit{I}\textsc{nit}, \\ \quad \mathit{mobiles}.\textit{I}\textsc{nit} \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \end{array}} \ (\ \mathit{thin}\ )}{\begin{array}{l} \mathit{MobilePhoneSystem} :: \textit{I}\textsc{nit};\ c_1, c_2 : \mathit{cells}.\mathit{cells} \mid \mathit{cells} \in \mathit{Cells}, \\ \quad \mathit{cells}.\textit{I}\textsc{nit} \wedge \mathit{mobiles}.\textit{I}\textsc{nit} \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \end{array}} \ (\ \wedge\vdash\ )}{\begin{array}{l} \mathit{MobilePhoneSystem} :: \textit{I}\textsc{nit};\ c_1, c_2 : \mathit{cells}.\mathit{cells} \mid \mathit{cells} \in \mathit{Cells} \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \end{array}} \ (\ \underline{\mathit{IntroduceInitPredicate}}\ )$$

Then we introduce *cells*' initial condition. (4)

$$\cfrac{\begin{array}{l} \mathit{MobilePhoneSystem} :: c_1, c_2 : \mathit{cells}.\mathit{cells} \mid \mathit{cells} \in \mathit{Cells} \\ \quad \forall\, c : \mathit{cells}.\mathit{cells} \bullet c.\textit{I}\textsc{nit} \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \end{array}}{\begin{array}{l} \mathit{MobilePhoneSystem} :: c_1, c_2 : \mathit{cells}.\mathit{cells} \mid \mathit{cells} \in \mathit{Cells}, \mathit{cells}.\textit{I}\textsc{nit} \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \end{array}} \ (\ \underline{\mathit{ExpandObjectInitPredicate}}\ )$$

We then deduce that c_1 is in its initial state. (5)

$$\cfrac{\cfrac{\begin{array}{l} \mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \mathit{cells} \in \mathit{Cells}, \\ \quad c_1.\textit{I}\textsc{nit} \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \end{array}}{\begin{array}{l} \mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \mathit{cells} \in \mathit{Cells}, \\ \quad \forall\, c : \mathit{cells}.\mathit{cells} \bullet c.\textit{I}\textsc{nit} \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \end{array}} \ (\ \underline{\mathit{ApplyForAllThin}}\ )}{\begin{array}{l} \mathit{MobilePhoneSystem} :: c_1, c_2 : \mathit{cells}.\mathit{cells} \mid \mathit{cells} \in \mathit{Cells}, \\ \quad \forall\, c : \mathit{cells}.\mathit{cells} \bullet c.\textit{I}\textsc{nit} \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \end{array}} \ (\ e\ )$$

where e is $x_1, \ldots, x_n : E \equiv x_1 : E;\ \ldots;\ x_n : E$

Next we introduce the property that *cells.cells* is a set of *Cell* objects. (6)

$$
\begin{array}{ll}
\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid c_1.\textsc{Init}, & \\
\quad \mathit{cells}.\mathit{cells} \in \mathbb{P}\,\mathit{Cell} \vdash & \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & (\ \underline{\mathit{FiniteSet}}\ ) \\
\hline
\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid c_1.\textsc{Init}, & \\
\quad \mathit{cells}.\mathit{cells} \in \mathbb{F}\,\mathit{Cell} \vdash & \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & (\ [n : s] \vdash\ ) \\
\hline
\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid c_1.\textsc{Init}, & \\
\quad [\mathit{cells}.\mathit{cells} : \mathbb{F}\,\mathit{Cell}] \vdash & \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & (\ \mathit{thin}\ ) \\
\hline
\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \mathit{cells} \in \mathit{Cells}, & \\
\quad c_1.\textsc{Init}, [\mathit{cells}.\mathit{cells} : \mathbb{F}\,\mathit{Cell}] \vdash & \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & (\ \underline{\mathit{IntroduceObjectLocalProperty}}\ ) \\
\hline
\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \mathit{cells} \in \mathit{Cells}, & \\
\quad c_1.\textsc{Init} \vdash & \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing &
\end{array}
$$

We then deduce that c_1 is an object of class *Cell*. (7)

$$
\begin{array}{ll}
\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid c_1.\textsc{Init}, & \\
\quad c_1 \in \mathit{Cell} \vdash & \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & (\ \underline{\mathit{ApplyToAntecedent}}(\mathit{powerset})\ ) \\
\hline
\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid c_1.\textsc{Init}, & \\
\quad \mathit{cells}.\mathit{cells} \in \mathbb{P}\,\mathit{Cell} \vdash & \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing &
\end{array}
$$

We introduce c_1's initial condition. (8)

$$
\begin{array}{ll}
\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid c_1 \in \mathit{Cell}, & \\
\quad c_1.\mathit{freq\_allocator}.\textsc{Init} \vdash & \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & (\ \mathit{thin}\ ) \\
\hline
\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid c_1 \in \mathit{Cell}, & \\
\quad c_1.\mathit{freq\_allocator}.\textsc{Init}, c_1.\mathit{seize\_unit}.\textsc{Init} \vdash & \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & (\ \wedge\vdash\ ) \\
\hline
\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid c_1 \in \mathit{Cell}, & \\
\quad c_1.\mathit{freq\_allocator}.\textsc{Init} \wedge c_1.\mathit{seize\_unit}.\textsc{Init} \vdash & \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & (\ \underline{\mathit{ExpandObjectInitPredicate}}\ ) \\
\hline
\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid c_1.\textsc{Init}, & \\
\quad c_1 \in \mathit{Cell} \vdash & \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing &
\end{array}
$$

We introduce the property that $c_1.\mathit{freq\_allocator}$ is an object of class $\mathit{FreqAllocator}$. (9)

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\begin{array}{l}\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\ \quad c_1.\mathit{freq\_allocator}.\textsc{Init}, c_1.\mathit{freq\_allocator} \in \mathit{FreqAllocator} \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing\end{array}}{\begin{array}{l}\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\ \quad c_1.\mathit{freq\_allocator}.\textsc{Init}, [c_1.\mathit{freq\_allocator} : \mathit{FreqAllocator}] \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing\end{array}}\ (\,[n : s] \vdash\,)}{\begin{array}{l}\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid c_1 \in \mathit{Cell}, \\ \quad c_1.\mathit{freq\_allocator}.\textsc{Init}, [c_1.\mathit{freq\_allocator} : \mathit{FreqAllocator}], \\ \quad [c_1.\mathit{seize\_unit} : \mathit{SeizeUnit}] \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing\end{array}}\ (\,\mathit{thin}\,)}{\begin{array}{l}\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid c_1 \in \mathit{Cell}, \\ \quad c_1.\mathit{freq\_allocator}.\textsc{Init}, \\ \quad [c_1.\mathit{freq\_allocator} : \mathit{FreqAllocator}] \wedge \\ \qquad [c_1.\mathit{seize\_unit} : \mathit{SeizeUnit}] \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing\end{array}}\ (\,\wedge\vdash\,)}{\begin{array}{l}\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid c_1 \in \mathit{Cell}, \\ \quad c_1.\mathit{freq\_allocator}.\textsc{Init}, \\ \quad [c_1.\mathit{freq\_allocator} : \mathit{FreqAllocator}; \\ \qquad c_1.\mathit{seize\_unit} : \mathit{SeizeUnit}] \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing\end{array}}\ (\,[D;\ D] \vdash\,)}{\begin{array}{l}\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid c_1 \in \mathit{Cell}, \\ \quad c_1.\mathit{freq\_allocator}.\textsc{Init} \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing\end{array}}\ (\,\underline{\mathit{IntroduceObjectLocalProperty}}\,)$$

Next we introduce $c_1.\mathit{freq\_allocator}$'s initial condition. (10)

$$\cfrac{\cfrac{\begin{array}{l}\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\ \quad c_1.\mathit{freq\_allocator.alloc} = \varnothing \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing\end{array}}{\begin{array}{l}\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\ \quad c_1.\mathit{freq\_allocator} \in \mathit{FreqAllocator}, c_1.\mathit{freq\_allocator.alloc} = \varnothing \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing\end{array}}\ (\,\mathit{thin}\,)}{\begin{array}{l}\mathit{MobilePhoneSystem} :: c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\ \quad c_1.\mathit{freq\_allocator}.\textsc{Init}, c_1.\mathit{freq\_allocator} \in \mathit{FreqAllocator} \vdash \\ \quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing\end{array}}\ (\,\underline{\mathit{ExpandObjectInitPredicate}}\,)$$

We then substitute the value of $c_1.\mathit{freq\_allocator}.\mathit{alloc}$ into the consequent.

$$\frac{}{\begin{array}{l} \mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \\ \quad c_1.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \vdash \\ \quad \varnothing = \varnothing \end{array}} \quad (\ \mathit{reflection}\ )$$

$$\frac{}{\begin{array}{l} \mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \\ \quad c_1.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \vdash \\ \quad \varnothing \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \end{array}} \quad (\ p(S \cap \varnothing) \equiv p(\varnothing)\ )$$

$$\frac{}{\begin{array}{l} \mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \\ \quad c_1.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \vdash \\ \quad \operatorname{dom} \varnothing \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \end{array}} \quad (\ p(\operatorname{dom} \varnothing) \equiv p(\varnothing)\ )$$

$$\frac{}{\begin{array}{l} \mathit{MobilePhoneSystem} :: c_1 : \mathit{cells}.\mathit{cells};\ c_2.\mathit{cells}.\mathit{cells} \mid \\ \quad c_1.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \vdash \\ \quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \end{array}} \quad (\ \underline{\mathit{ApplyToConsequent}}(\mathit{ApplyLeibniz})\ )$$

Appendix B

The following proof follows the outline for the *Page* operation given in Section 3 and uses tactics defined in Section 4 as well as the rule for alias control defined in Section 5. References to proof steps in the informal proof are given in parentheses. For example, (1) refers to proof step 1.

We begin by simplifying the consequent. This is done using the rules for implies and universal quantification elimination. (1)

$$
\begin{array}{ll}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells.cells} \mid & \\
\quad \forall\, c_1, c_2 : \mathit{cells.cells} \bullet c_1 \neq c_2 \Rightarrow & \\
\qquad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \emptyset, & \\
\qquad c_1 \neq c_2 \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \emptyset & \\
\hline
 & (\vdash\Rightarrow) \\
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells.cells} \mid & \\
\quad \forall\, c_1, c_2 : \mathit{cells.cells} \bullet c_1 \neq c_2 \Rightarrow & \\
\qquad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \emptyset \vdash & \\
\quad c_1 \neq c_2 \Rightarrow & \\
\qquad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \emptyset & \\
\hline
 & (\vdash\forall) \\
\mathit{MobilePhoneSystem} :: \mathit{Page} \mid \forall\, c_1, c_2 : \mathit{cells.cells} \bullet c_1 \neq c_2 \Rightarrow & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \emptyset \vdash & \\
\quad \forall\, c_1, c_2 : \mathit{cells.cells} \bullet c_1 \neq c_2 \Rightarrow & \\
\qquad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \emptyset & \\
\hline
 & (\vdash\Rightarrow) \\
\mathit{MobilePhoneSystem} :: \mathit{Page} \mid \vdash \forall\, c_1, c_2 : \mathit{cells.cells} \bullet c_1 \neq c_2 \Rightarrow & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \emptyset & \\
\quad \Rightarrow & \\
\quad \forall\, c_1, c_2 : \mathit{cells.cells} \bullet c_1 \neq c_2 \Rightarrow & \\
\qquad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \emptyset &
\end{array}
$$

We then deduce that the property holds for c_1 and c_2 before the operation. (2)

$$
\begin{array}{ll}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells.cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \emptyset \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \emptyset & \\
\hline
 & (\ \underline{\mathit{ApplyImpliesThin}}\ ) \\
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells.cells} \mid & \\
\quad c_1 \neq c_2 \Rightarrow & \\
\qquad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \emptyset, & \\
\quad c_1 \neq c_2 \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \emptyset & \\
\hline
 & (\ \underline{\mathit{ApplyForAllThin}}\ ) \\
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells.cells} \mid & \\
\quad \forall\, c_1, c_2 : \mathit{cells.cells} \bullet c_1 \neq c_2 \Rightarrow & \\
\qquad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \emptyset, & \\
\quad c_1 \neq c_2 \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \emptyset &
\end{array}
$$

We then introduce the property that *cells* is an object of class *Cells*. (3)

$$
\begin{array}{ll}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells}.\mathit{cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & \\
\quad \mathit{cells} \in \mathit{Cells} \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing & \\
\hline
 & (\ [n : s] \vdash\ ) \\
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells}.\mathit{cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & \\
\quad [\mathit{cells} : \mathit{Cells}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing & \\
\hline
 & (\ \mathit{thin}\ ) \\
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells}.\mathit{cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & \\
\quad [\mathit{cells} : \mathit{Cells}], [\mathit{mobiles} : \mathit{Mobiles}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing & \\
\hline
 & (\ \wedge\vdash\ ) \\
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells}.\mathit{cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & \\
\quad [\mathit{cells} : \mathit{Cells}] \wedge [\mathit{mobiles} : \mathit{Mobiles}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing & \\
\hline
 & (\ [D;\ D] \vdash\ ) \\
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells}.\mathit{cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing & \\
\quad [\mathit{cells} : \mathit{Cells};\ \mathit{mobiles} : \mathit{Mobiles}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing & \\
\hline
 & (\ \mathit{LocalDefs}\ ) \\
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells}.\mathit{cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing &
\end{array}
$$

Next we introduce the property that *cells.cells* is a set of *Cell* objects. (4)

$$
\begin{array}{ll}
\quad \mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells.cells} \mid & \\
\qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\qquad \mathit{cells.cells} \in \mathbb{P}\, \mathit{Cell} \vdash & \\
\qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc}' = \varnothing & \\
\hline
\quad \mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells.cells} \mid & (\ \underline{\mathit{FiniteSet}}\ ) \\
\qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\qquad \mathit{cells.cells} \in \mathbb{F}\, \mathit{Cell} \vdash & \\
\qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc}' = \varnothing & \\
\hline
\quad \mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells.cells} \mid & (\ [n : s] \vdash\ ) \\
\qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\qquad [\mathit{cells.cells} : \mathbb{F}\, \mathit{Cell}] \vdash & \\
\qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc}' = \varnothing & \\
\hline
\quad \mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells.cells} \mid & (\ \mathit{thin}\ ) \\
\qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\qquad \mathit{cells} \in \mathit{Cells}, [\mathit{cells.cells} : \mathbb{F}\, \mathit{Cell}] \vdash & \\
\qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc}' = \varnothing & \\
\hline
\quad \mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells.cells} \mid & (\ \underline{\mathit{IntroduceObjectLocalProperty}}\ ) \\
\qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc} = \varnothing & \\
\qquad \mathit{cells} \in \mathit{Cells} \vdash & \\
\qquad \operatorname{dom} c_1.\mathit{freq\_allocator.alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator.alloc}' = \varnothing &
\end{array}
$$

We then deduce that c_1 and c_2 are objects of class Cell. (5)

$$
\begin{array}{ll}
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing \\
\quad c_1 \in \mathit{Cell}, c_2 \in \mathit{Cell} \vdash \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc}' = \varnothing
\end{array} & \\
\hline
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing \\
\quad c_1 \in \mathit{Cell}, \mathit{cells.cells} \in \mathbb{P}\, \mathit{Cell} \vdash \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc}' = \varnothing
\end{array} & (\ \underline{\mathit{ApplyToAntecedent}(\mathit{powerset})}\ ) \\
\hline
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing \\
\quad \mathit{cells.cells} \in \mathbb{P}\, \mathit{Cell}, \mathit{cells.cells} \in \mathbb{P}\, \mathit{Cell} \vdash \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc}' = \varnothing
\end{array} & (\ \underline{\mathit{ApplyToAntecedent}(\mathit{powerset})}\ ) \\
\hline
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing \\
\quad \mathit{cells.cells} \in \mathbb{P}\, \mathit{Cell} \wedge \mathit{cells.cells} \in \mathbb{P}\, \mathit{Cell} \vdash \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc}' = \varnothing
\end{array} & (\ \wedge\!\vdash\ ) \\
\hline
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing \\
\quad \mathit{cells.cells} \in \mathbb{P}\, \mathit{Cell} \vdash \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc}' = \varnothing
\end{array} & (\ p \equiv p \wedge p\ ) \\
\hline
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1, c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc} = \varnothing \\
\quad \mathit{cells.cells} \in \mathbb{P}\, \mathit{Cell} \vdash \\
\quad \mathrm{dom}\, c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\, c_2.\mathit{freq\_allocator.alloc}' = \varnothing
\end{array} & (\ e\ )
\end{array}
$$

where e is $x_1, \ldots, x_n : E \equiv x_1 : E;\ \ldots;\ x_n : E$

We introduce the property that $c_1.\mathit{freq\_allocator}$ is an object of class $\mathit{FreqAllocator}$. (6)

$$\begin{array}{ll}
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc} = \varnothing, \\
\quad c_2 \in \mathit{Cell}, c_1.\mathit{freq_allocator} \in \mathit{FreqAllocator} \vdash \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc}' = \varnothing
\end{array} & \\
\hline
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc} = \varnothing, \\
\quad c_2 \in \mathit{Cell}, [c_1.\mathit{freq_allocator} : \mathit{FreqAllocator}] \vdash \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc}' = \varnothing
\end{array} & (\ [n : s] \vdash\ ) \\
\hline
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc} = \varnothing \\
\quad c_2 \in \mathit{Cell}, [c_1.\mathit{freq_allocator} : \mathit{FreqAllocator}], \\
\quad [c_1.\mathit{seize_unit} : \mathit{SeizeUnit}] \vdash \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc}' = \varnothing
\end{array} & (\ \mathit{thin}\ ) \\
\hline
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc} = \varnothing, \\
\quad c_2 \in \mathit{Cell}, \\
\quad [c_1.\mathit{freq_allocator} : \mathit{FreqAllocator}] \wedge [c_1.\mathit{seize_unit} : \mathit{SeizeUnit}] \vdash \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc}' = \varnothing
\end{array} & (\ \wedge\vdash\ ) \\
\hline
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc} = \varnothing, \\
\quad c_2 \in \mathit{Cell}, \\
\quad [c_1.\mathit{freq_allocator} : \mathit{FreqAllocator};\ c_1.\mathit{seize_unit} : \mathit{SeizeUnit}] \vdash \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc}' = \varnothing
\end{array} & (\ [D;\ D] \vdash\ ) \\
\hline
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc} = \varnothing, \\
\quad c_1 \in \mathit{Cell}, c_2 \in \mathit{Cell}, \\
\quad [c_1.\mathit{freq_allocator} : \mathit{FreqAllocator};\ c_1.\mathit{seize_unit} : \mathit{SeizeUnit}] \vdash \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc}' = \varnothing
\end{array} & (\ \mathit{thin}\ ) \\
\hline
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc} = \varnothing \\
\quad c_1 \in \mathit{Cell}, c_2 \in \mathit{Cell} \vdash \\
\quad \mathrm{dom}\ c_1.\mathit{freq_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq_allocator.alloc}' = \varnothing
\end{array} & (\ \underline{\mathit{IntroduceObjectLocalProperty}}\ )
\end{array}$$

We introduce the property that $c_2.\mathit{freq\_allocator}$ is an object of class $\mathit{FreqAllocator}$. (7)

$$
\begin{array}{l l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing, & \\
\quad c_1.\mathit{freq\_allocator} \in \mathit{FreqAllocator}, & \\
\quad c_2.\mathit{freq\_allocator} \in \mathit{FreqAllocator} \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \varnothing & (\ [n : s] \vdash\ ) \\
\hline
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing, & \\
\quad c_1.\mathit{freq\_allocator} \in \mathit{FreqAllocator}, & \\
\quad [c_2.\mathit{freq\_allocator} : \mathit{FreqAllocator}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \varnothing & (\ \mathit{thin}\ ) \\
\hline
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing, & \\
\quad c_1.\mathit{freq\_allocator} \in \mathit{FreqAllocator}, & \\
\quad [c_2.\mathit{freq\_allocator} : \mathit{FreqAllocator}], [c_2.\mathit{seize\_unit} : \mathit{SeizeUnit}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \varnothing & (\ \wedge\vdash\ ) \\
\hline
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing, & \\
\quad c_1.\mathit{freq\_allocator} \in \mathit{FreqAllocator}, & \\
\quad [c_2.\mathit{freq\_allocator} : \mathit{FreqAllocator}] \wedge [c_2.\mathit{seize\_unit} : \mathit{SeizeUnit}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \varnothing & (\ [D;\ D] \vdash\ ) \\
\hline
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing, & \\
\quad c_1.\mathit{freq\_allocator} \in \mathit{FreqAllocator}, & \\
\quad [c_2.\mathit{freq\_allocator} : \mathit{FreqAllocator};\ c_2.\mathit{seize\_unit} : \mathit{SeizeUnit}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \varnothing & (\ \mathit{thin}\ ) \\
\hline
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing, & \\
\quad c_2 \in \mathit{Cell}, c_1.\mathit{freq\_allocator} \in \mathit{FreqAllocator}, & \\
\quad [c_2.\mathit{freq\_allocator} : \mathit{FreqAllocator};\ c_2.\mathit{seize\_unit} : \mathit{SeizeUnit}] \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \varnothing & (\ \underline{\mathit{IntroduceObjectLocalProperty}}\ ) \\
\hline
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells.cells};\ c_2 : \mathit{cells.cells} \mid & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc} \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc} = \varnothing, & \\
\quad c_2 \in \mathit{Cell}, c_1.\mathit{freq\_allocator} \in \mathit{FreqAllocator} \vdash & \\
\quad \mathrm{dom}\ c_1.\mathit{freq\_allocator.alloc}' \cap \mathrm{dom}\ c_2.\mathit{freq\_allocator.alloc}' = \varnothing &
\end{array}
$$

We then introduce the fact that $c_1.freq\_allocator.alloc$ is unchanged. (8)

Since $\chi(MobilePhoneSystem, Page) = \{(cells, Cells), (c, Cell)\}$, the third predicate of the proviso is true. The other predicates of the proviso follow directly from the specification.

$$\dfrac{\dfrac{\begin{array}{l} MobilePhoneSystem :: Page;\ c_1 : cells.cells;\ c_2 : cells.cells \mid \\ \quad \mathrm{dom}\ c_1.freq\_allocator.alloc \cap \mathrm{dom}\ c_2.freq\_allocator.alloc = \varnothing, \\ \quad c_2.freq\_allocator \in FreqAllocator, \\ \quad c_1.freq\_allocator.alloc' = c_1.freq\_allocator.alloc \vdash \\ \quad \mathrm{dom}\ c_1.freq\_allocator.alloc' \cap \mathrm{dom}\ c_2.freq\_allocator.alloc' = \varnothing \end{array}}{\begin{array}{l} MobilePhoneSystem :: Page;\ c_1 : cells.cells;\ c_2 : cells.cells \mid \\ \quad \mathrm{dom}\ c_1.freq\_allocator.alloc \cap \mathrm{dom}\ c_2.freq\_allocator.alloc = \varnothing, \\ \quad c_1.freq\_allocator \in FreqAllocator, \\ \quad c_2.freq\_allocator \in FreqAllocator, \\ \quad c_1.freq\_allocator.alloc' = c_1.freq\_allocator.alloc \vdash \\ \quad \mathrm{dom}\ c_1.freq\_allocator.alloc' \cap \mathrm{dom}\ c_2.freq\_allocator.alloc' = \varnothing \end{array}} (\ thin\ )}{\begin{array}{l} MobilePhoneSystem :: Page;\ c_1 : cells.cells;\ c_2 : cells.cells \mid \\ \quad \mathrm{dom}\ c_1.freq\_allocator.alloc \cap \mathrm{dom}\ c_2.freq\_allocator.alloc = \varnothing, \\ \quad c_1.freq\_allocator \in FreqAllocator, \\ \quad c_2.freq\_allocator \in FreqAllocator \vdash \\ \quad \mathrm{dom}\ c_1.freq\_allocator.alloc' \cap \mathrm{dom}\ c_2.freq\_allocator.alloc' = \varnothing \end{array}} (\ UnchangedObject\ )\ [\ q\ ]$$

where $q \equiv \textsc{Sys} \in MobilePhoneSystem$
$\iota^*(FreqAllocator) = \varnothing$
$\chi(MobilePhoneSystem, Page) \rhd \{FreqAllocator\} = \varnothing$
$\alpha_1(FreqAllocator) = \{alloc\}$

Similarly, we introduce the fact that $c_2.\mathit{freq\_allocator}.\mathit{alloc}$ is unchanged. (9)

$$
\begin{array}{ll}
\begin{array}{l}
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \\
\quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing, \\
\quad c_1.\mathit{freq\_allocator}.\mathit{alloc}' = c_1.\mathit{freq\_allocator}.\mathit{alloc}, \\
\quad c_2.\mathit{freq\_allocator}.\mathit{alloc}' = c_2.\mathit{freq\_allocator}.\mathit{alloc} \vdash \\
\quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing \\
\hline
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \\
\quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing, \\
\quad c_2.\mathit{freq\_allocator} \in \mathit{FreqAllocator}, \\
\quad c_1.\mathit{freq\_allocator}.\mathit{alloc}' = c_1.\mathit{freq\_allocator}.\mathit{alloc}, \\
\quad c_2.\mathit{freq\_allocator}.\mathit{alloc}' = c_2.\mathit{freq\_allocator}.\mathit{alloc} \vdash \\
\quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing \\
\hline
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \\
\quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing, \\
\quad c_2.\mathit{freq\_allocator} \in \mathit{FreqAllocator}, \\
\quad c_1.\mathit{freq\_allocator}.\mathit{alloc}' = c_1.\mathit{freq\_allocator}.\mathit{alloc} \vdash \\
\quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing
\end{array}
&
\begin{array}{l}
(\ \mathit{thin}\ ) \\[4em]
(\ \mathit{UnchangedObject}\ )\ [\ q\ ]
\end{array}
\end{array}
$$

$$
\begin{array}{ll}
\text{where } q \equiv \textsc{Sys} \in \mathit{MobilePhoneSystem} & \\
\qquad \iota^*(\mathit{FreqAllocator}) = \varnothing & \\
\qquad \chi(\mathit{MobilePhoneSystem}, \mathit{Page}) \rhd \{\mathit{FreqAllocator}\} \subseteq \varnothing & \\
\qquad \alpha_1(\mathit{FreqAllocator}) = \{\mathit{alloc}\} &
\end{array}
$$

We then replace $c_1.\mathit{freq\_allocator}.\mathit{alloc}$ and $c_2.\mathit{freq\_allocator}.\mathit{alloc}$ in the antecedent with $c_1.\mathit{freq\_allocator}.\mathit{alloc}'$ and $c_2.\mathit{freq\_allocator}.\mathit{alloc}'$ respectively. (10)

$$
\begin{array}{ll}
\begin{array}{l}
\hline
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells}, \mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \\
\quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing \vdash \\
\quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing \\
\hline
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \\
\quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing, \\
\quad c_2.\mathit{freq\_allocator}.\mathit{alloc}' = c_2.\mathit{freq\_allocator}.\mathit{alloc} \vdash \\
\quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing \\
\hline
\mathit{MobilePhoneSystem} :: \mathit{Page};\ c_1 : \mathit{cells}.\mathit{cells};\ c_2 : \mathit{cells}.\mathit{cells} \mid \\
\quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc} \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc} = \varnothing, \\
\quad c_1.\mathit{freq\_allocator}.\mathit{alloc}' = c_1.\mathit{freq\_allocator}.\mathit{alloc}, \\
\quad c_2.\mathit{freq\_allocator}.\mathit{alloc}' = c_2.\mathit{freq\_allocator}.\mathit{alloc} \vdash \\
\quad \operatorname{dom} c_1.\mathit{freq\_allocator}.\mathit{alloc}' \cap \operatorname{dom} c_2.\mathit{freq\_allocator}.\mathit{alloc}' = \varnothing
\end{array}
&
\begin{array}{l}
(\ \mathit{assumption}\ ) \\[3em]
(\ \underline{\mathit{ApplyLeibnizThin}}\ ) \\[3em]
(\ \underline{\mathit{ApplyLeibnizThin}}\ )
\end{array}
\end{array}
$$