Liveness Properties under Fairness Assumptions and Semantics of Systems in event B

Hector Ruiz Barradas

UNIVERSITÉ JOSEPH FOURIER (GRENOBLE I)

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER

Spécialité : INFORMATIQUE

École Doctorale Mathématiques Informatique Sciences et Technologies de L'Information

présentée et soutenue publiquement
par

Héctor Ruíz Barradas

le 22 décembre 2006

Propriétés de vivacité sous conditions d'équité et sémantique des systèmes d'événements avec la méthode B

JURY

Rapporteurs	M. Dominique Méry	Professeur Université Henri Poincaré
	M. Gérard Padiou	Professeur ENSEEIHT
Examinateur	M. Farid Ouabdesselam	Professeru Université Joseph Fourier
Directeur de thèse	M. Didier Bert	CNRS

Thèse préparée au sein du Laboratoire LSR
"Logiciels Systèmes Réseaux´'

Remerciements

Je tiens à remercier à M. Dominique Méry et M. Gérard Padiou d'avoir accepté d'être les rapporteurs de cette travail de recherche.

Je tiens tout spécialement remercier à M. Farid Ouabdesselam, président de mon jury de thèse et directeur du laboratoire L.S.R., pour ses conseils pertinents qu'il m'a dispensés tout au long de mon travail.

J'exprime ma plus sincère gratitude à M. Didier Bert, mon directeur de thèse, pour l'intérêt qu'il a attribué à mon travail de recherche, pour les critiques toujours constructives et pour les nombreuses corrections qu'il a apportées à toute ma production écrite.

J'adresse de plus mes remerciements à l'*Universidad Autónoma Metropolitana Azcapotzalco* pour avoir financé ce travail. Tout particulièrement, je remercie la *División de Ciencias Básicas e Ingeniería* pour m'avoir accordé une bourse et le *Departamento de Electrónica*, dont je suis enseignant chercheur, pour l'aide efficace qu'il m'a apportée dans l'accomplissement des formalités nécessaires à ce travail de recherche. Dans ce cadre, je remercie tout spécialement à M. Armando Jimenez Flores.

Je remercie tout particulièrement M. Dominique Decouchant pour son amitié et pour tous les services qui m'a procurés tout au long de ces années. De la même manière je remercie Mmes Nicole Fréry, Marie Catherine Lévêque, Hanka Zeman et Françoise Jalabert ainsi que M. Michel Bands.

Je remercie de tout mon coeur mon épouse Cecilia pour son soutien et ses encouragements dans les moments difficiles, ainsi que pour sa solidarité et son engagement dans toutes les démarches de la vie quotidienne. Sans son aide et son amour, cette thèse n'aurait pu être réalisée.

Je remercie mes fils : Héctor Tomas, Jorge Alfredo, Luis Alberto et Cesar Jesus. Leur caractères tranquilles et leurs comportements déjà aussi matures m'ont beaucoup aidé dans mon travail. En outre, toute ma famille a été une source d'encouragements permanente.

Je tiens à remercier à Nelly Gladys Centurión Prado et à Manuel Chayan Mendoza pour tout leur soutien moral essentiel.

Finalement je remercie mon père Héctor Ruiz Vazques ainsi que ma mère Trinidad Barradas Jimenez, pour m'avoir inculqué le sens du travail et les valeurs essentielles de la responsabilité et de l'amour. Mes remerciements vont aussi à ma soeur Dulce María et à mes frères Alfredo et Jorge ; les actions qu'ils ont développées et la solidarité qu'ils ont tissée autour de mes parents m'ont procuré la sérénité nécessaire dans la réalisation de ma tâche.

ABSTRACT

This thesis proposes an approach to the specification and proof of liveness properties under fairness assumptions in B event system and presents a semantics for these properties founded on fixpoints of set transformers. The proposal uses a UNITY-like logic to specify and verify liveness properties under minimal progress and weak fairness assumptions and presents rules to preserve these properties under refinement. The fixpoint semantics allows us to make the notion of *reachability* under fairness assumptions equivalent to the one of *termination* of iteration of events. Soundness and completeness of rules to derive liveness properties are proved, thanks to this equivalence. Moreover, the fixpoint semantics provides the foundations to prove soundness of both the rules for verification of liveness properties and the rules for the preservation of liveness under refinement.

RÉSUMÉ

Cette thèse propose une approche à la spécification et preuve des propriétés de vivacité avec hypothèses d'équité en B événementiel et présente une sémantique pour ces propriétés fondée sur de points fixes de transformateurs d'ensembles. La proposition utilise une logique de programmation issue de la logique UNITY pour spécifier et vérifier des propriétés de vivacité sous des hypothèses de progrès minimal et d'équité faible et présente des règles pour préserver ces propriétés dans les raffinements. La sémantique de points fixes nous permet de faire équivalentes les notions d'*atteignabilité* sous les hypothèses d'équité et de *terminaison* de l'itération d'événements. Cela nous permet de prouver la correction et la complétude des règles permettant la dérivation des propriétés de vivacité. En outre, cela donne les fondements pour prouver la correction des règles permettant la vérification des propriétés de vivacité et des règles permettant la préservation de la vivacité sous raffinement.

Contents

Glossary of New Notations

Chapter 1

Introduction

1.1 Version française

Les systèmes d'actions ou d'événements sont des abstractions utiles pour modéliser des systèmes discrets. Le modèle de calcul de ces abstractions considère une collection d'événements (actions) qui transforment l'état d'une manière atomique et non déterministe. Afin d'assurer l'exécution de certains événements, malgré le non déterminisme, quelques formalismes introduisent des hypothèses d'équité qui aident à élever le niveau d'abstraction d'une spécification parallèle ou distribuée. Dans ce contexte, l'équité signifie que si une certaine opération est possible, alors le système doit fatalement l'exécuter [41]. La transformation de l'état peut être observée par les traces d'états, qui décrivent le comportement du système ; cela permet la spécification d'un système à un haut niveau d'abstraction. La mise en œuvre d'un système d'actions se fonde dans la notion de pas de raffinement. Dans cette notion, les structures de données abstraites sont mises en œuvre par des structures qui sont souvent utilisées dans les langages de programmation, et les transformations atomiques de l'état sont faites par des algorithmes (séquence d'instructions) qui travaillent à un bas niveau de granularité.

Plusieurs formalismes, avec différentes approches, ont été proposé pour modéliser des systèmes d'actions. Cependant, nous pouvons approximativement classer ces approches en trois catégories : logiques, algorithmiques et hybrides.

Dans l'approche logique, la spécification et le raffinement d'un système est énoncé en termes de formules, dans une certaine logique, qui spécifient deux classes de propriétés : sécurité *et* vivacité. *Les propriétés de vivacité énoncent que quelque chose de mauvais n'arrive jamais, c'est-à-dire que le programme n'entre jamais dans un état inacceptable, tandis que les propriétés de vivacité affirment que quelque chose de bien arrive fatalement, c'est à dire que le programme entre fatalement dans un état souhaitable. Les propriétés de vivacité sont spécifiées par les différentes hypothèses d'équité. Le système de preuve de la logique est utilisé pour démontrer la correction du modèle. Comme un exemple de formalisme dans cette catégorie, nous pouvons citer* TLA *[41].*

Dans l'approche algorithmique, la spécification et le raffinement d'un système est donné comme une notation de programmation. Les propriétés de vivacité sont implicitement spécifiées par le comportement du système d'actions, qui est décrit par ses traces d'états. Comme un exemple de cette classe d'approches, nous pouvons citer le Système d'Actions de Back [10]. Dans ce formalisme, l'équité dans l'exécution d'actions est spécifiée par des itérations équitables. Le calcul de raffinements est utilisé pour assurer la correction des raffinements.

Dans l'approche hybride, l'aspect algorithmique d'un système est donné comme une notation de programmation, et ses propriétés sont décrites dans une certaine logique. Cependant, dans quelle mesure appliquons nous ces approches dans un développement ? Cela dépend de la méthode formelle. En UNITY *[21], par exemple, la spécification et le raffinement d'un système sont faits d'une manière purement logique, par la spécification de propriétés de sûreté et de vivacité dans une logique temporelle. La correction des raffinements, comme dans* TLA*, est prouvée par l'implication logique entre une spécification et son raffinement. Le dernier pas d'un raffinement permet la dérivation d'un programme, dans un langage de programmation non déterministe, qui peut être mis en œuvre par un traduction informelle, dans différentes architectures parallèles. D'autre part, dans le* B *événementiel [7], la spécification et le raffinement de systèmes sont données comme une notation de programmation, alors que certaines formes des propriétés de vivacité peuvent être explicitement énoncées dans les spécifications et raffinements. La correction des raffinements est prouvée par des règles qui peuvent être comparées à celles du calcul des raffinements.*

Par notre expérience, nous pensons que les approches logiques et algorithmiques ont besoin de quelques étapes qui ne sont pas incluses dans ses méthodologies. Les systèmes au niveau de mise en œuvre ne sont pas nécessairement des formules logiques, et l'approche logique, telle qu'elle est pratiquée par TLA*, a besoin des quelques pas pour traduire la description logique d'un système dans une notation pratique, qui permet sa mise en œuvre dans des ordinateurs conventionnels. D'autre part, l'approche algorithmique dans le style des Système d'Actions de Back, n'a pas des moyens de spécifier des propriétés de sûreté ou de vivacité, souvent énoncées en logique temporelle, qui permettent de faire des raisonnements sur les propriétés du système. Pour ces raisons, nous sommes intéressés dans les approches hybrides où l'aspect algorithmique d'un système et ses propriétés sont développées en même temps ; cependant, les approches proposées par* UNITY *ou* B *événementiel peuvent être améliorées.*

Dans l'approche UNITY*, la dérivation de programmes après le dernier pas de raffinement n'est pas systématique. Pour cette raison, la mise en œuvre d'un système perd son traitement rigoureux. Dans le* B *événementiel, des systèmes peuvent être développés d'une manière stricte de la spécification à la mise en œuvre. Cependant, la spécification et preuve de propriétés de vivacité nous semble restrictive : la construction utilisée pour la spécification des propriétés de vivacité ne prend pas en compte des hypothèses d'équité, et la seule manière de vérifier ces propriétés est par des règles qui ressemblent à des règles de terminaison d'itération.*

La première contribution principale de cette thèse est une proposition pour la conception de systèmes d'actions fondée sur le B *événementiel. La proposition prend en compte des hypothèses d'équité et utilise une logique comme celle d'*UNITY *pour spécifier et prouver des propriétés de vivacité. Dans notre travail, nous considérons des modalités du type* P leads-to Q*, où* P *et* Q *sont des prédicats dans l'espace d'état du système, avec la signification informelle suivante : "lorsque le système arrive dans un état qui satisfait* P*, il* atteint *fatalement un état qui satisfait* Q*". Nous proposons des règles de preuve pour vérifier des modalités sous les hypothèses de progrès minimal et d'équité faible, qui sont fondées sur le calcul des plus faibles préconditions. En outre, nous proposons des règles de preuve pour garantir que les propriétés de vivacité sont préservées par raffinement.*

La sémantique des systèmes d'actions est en général énoncé par des traces d'états. Les traces d'états des systèmes de transitions imposent un raisonnement opérationnel sur le comportement d'un système. Cependant, ce comportement opérationnel peut être caché par l'utilisation de la logique temporelle pour spécifier des propriétés de vivacité et de sûreté. La sémantique des formules temporelles est donnée par des traces d'états du système de transi-

tions. Le système de preuve de ces logiques permet la dérivation d'autres propriétés à partir des propriétés déjà prouvées, uniquement par des calculs symboliques. La cohérence et complétude de la logique sont établies par des preuves qui mettent en relation des formules logiques avec des assertions sur les traces d'états. A ce point, nous revenons au raisonnement opérationnel sur le système de transitions.

*Dans cette thèse nous proposons une possibilité plus abstraite de la sémantique des systèmes d'actions fondée sur les points fixes de transformateurs d'ensembles (ou prédicats). Ces points fixes dénotent le plus grand sous-ensemble d´états où l'itération d'événements termine sous les hypothèses d'équité considérées. Notre approche n'est pas une question de goût ou d'habitude, mais cela nous permet de montrer que les notions d'*atteignabilité *sous des hypothèses d'équité et de* terminaison *d'itération d'événements sont équivalentes. L'équivalence nous permet de prouver la cohérence et la complétude des règles proposées pour dériver des propriétés* leads-to*, et en outre, elle nous donne les fondements pour prouver la cohérence des règles pour la préservation de la vivacité dans les raffinements. Cela est la deuxième contribution principale de cette thèse, dans la ligne des travaux sur la sémantique dénotationnelle développée dans les années 60. Nous pensons qu'il est important de donner des bases communes aux différentes branches de l'informatique, afin de unifier les fondations et la compréhension de notre discipline.*

La thèse est divisée en sept chapitres. Dans le chapitre 2, nous donnons un aperçu des formalismes classiques pour la spécification des systèmes d'actions : UNITY, TLA*, systèmes d'actions de Back et* B *événementiel. Afin de mieux comprendre les similarités et les différences entre ces formalismes, nous traitons un exemple de spécification et raffinement d'un système abstrait producteur-consommateur dans tous ces formalismes.*

Le chapitre 3 présente notre approche de l'intégration d'une logique au style UNITY *pour la spécification et preuve des propriétés de vivacité sous les hypothèses de progrès minimal et d'équité faible. Nous présentons des règles de preuve pour la vérification et pour la préservation dans les raffinements des propriétés de vivacité. L'application de ces règles est illustrée par plusieurs exemples qui modèlent différents algorithmes d'exclusion mutuelle.*

Le chapitre 4 introduit la sémantique de nos formules en termes des points fixes de transformateurs d'ensembles. Dans ce chapitre nous définissons des transformateurs d'ensembles qui modélisent l'itération d'événements sous les hypothèses de progrès minimal et d'équité faible. Ces transformateurs d'ensembles permettent la définition de la relation de terminaison *qui est utilisée pour interpréter des propriétés* leads-to*. D'autre part, l'ensemble des propriétés de vivacité d'un système est dénotée par une autre relation entre sous-ensembles d'états nommée* atteignabilité*. La cohérence et la complétude relative des règles pour* leads-to *sont prouvées par l'utilisation de l'égalité entre ces relations.*

Dans le chapitre 5 nous prouvons la cohérence des règles présentées dans le chapitre 3 pour garantir la préservation des propriétés de vivacité dans les raffinements de systèmes B *événementiel. Dans le chapitre 6 nous développons un exemple plus important d'exclusion mutuelle, sous l'hypothèse d'équité faible, dont les raffinements sont prévus pour être mis en œuvre sur des environnements centralisés ou distribués.*

Finalement, dans la chapitre 7 nous donnons nos conclusions, nous comparons notre travail et nous analysons le travail futur.

1.2 English Version

Action or event systems are useful abstractions to model discrete systems. The computational model of these abstractions considers a collection of events (actions) transforming the state in an atomic and nondeterministic way. In order to ensure the execution of certain events, in spite of nondeterminism, some formalisms introduce fairness assumptions which help to increase the abstraction level of a parallel or distributed system specification. In this context, fairness means that if a certain operation is possible, then the system must eventually execute it [41]. The transformation of the state can be observed by state traces, describing the behavior of the system, allowing its specification at a high level of abstraction. The implementation of action systems is founded on stepwise refinements where abstract data structures are implemented by structures commonly used in programming languages and the atomic transformations of the state are made by algorithms (statement sequences) working at low level of granularity.

Many formalisms, with different approaches, have been proposed to model action systems. However, we can roughly classify the approaches of these formalisms in three categories: logical, algorithmic and hybrid.

In the logical approach, the specification and refinement of a system is stated in terms of formulas in a certain logic specifying two kinds of properties: *safety* and *liveness*. Safety properties state that something bad never happens –that is, that the program never enters an unacceptable state and liveness properties assert that something good eventually does happen – that is, that the program eventually enters a desirable state. Liveness is specified by different fairness assumptions. The associated proof system of the logic is used to demonstrate the correctness of the design. As an example of formalism in this category, we can cite TLA [41].

In the algorithmic approach, the specification and refinement of a system is expressed in a programming-like notation. Safety and liveness properties are implicitly specified by the behavior of the action system, described by its state traces. As an example of this kind of approach, we can cite Back's Action System formalism [10]. In this formalism, fairness in the execution of actions is specified by fair iteration. The refinement calculus is used to ensure the correctness of refinements.

In the hybrid approach, a programming-like notation is used to describe the algorithmic aspect of the system and its properties are described in a certain logic. However, to what extent one applies these approaches in a development? It depends on the formal method. In UNITY [21], for example, the specification and refinement of a system is made in purely logical way, specifying safety and liveness properties in a temporal logic. Correctness of refinements, as in TLA, is proved by logical implication between a specification and its refinement. The last step of a refinement allows the derivation of a program in a nondeterministic programming language which can be implemented, by an informal translation, in several types of parallel architectures. On another hand, in the event B formalism [7], the specification and refinement of systems are expressed in a programming-like notation whereas certain forms of safety and liveness properties can be explicitly stated in these specifications or refinements. Correctness of refinements is proved by rules which can be compared to those in the refinement calculus.

In our opinion, based in our experience, the algorithmic and logical approaches need some steps not included in their methodologies. Systems, at the implementation level, are not necessarily logical formulas, and the logical approach, as practiced by TLA, needs some steps to translate the logical description of a system into a practical notation, allowing its implementation in conventional computers. On another hand, the algorithmic approach in the style of Back's Action System, has no means to assert safety or liveness properties often

specified in temporal logic and then deduce facts about the system. For these reasons, we are interested in the hybrid approach to action or event systems, where the algorithmic aspect of a system and its properties are developed at the same time; however, the approaches of UNITY and event B may be improved.

In the UNITY approach, the derivation of programs after the last refinement step is not systematic. Therefore, the implementation of a system looses its rigorous treatment. In event B, some systems can be developed in a rigorous way from the specification to its implementation; nevertheless, the specification and proof of liveness properties seem to us restrictive: The construct used to specify liveness properties does not take into account fairness assumptions and the only way to verify these properties is by rules looking like termination of iteration.

The first main contribution of this thesis is a proposal to design action systems founded on the event B formalism, where we take into account fairness assumptions and we use an UNITY-like logic to specify and prove liveness properties. In our work, we consider modalities of type P *leads-to* Q, where P and Q are predicates on the state space of a system, with the informal meaning: "when the system arrives into a state satisfying P, eventually it *reaches* a state satisfying Q". We propose proof rules to verify modalities under minimal progress and weak fairness assumptions, founded in the weakest precondition calculus. Moreover, we propose proof rules to guarantee that liveness properties are preserved under refinement.

Semantics of action systems is generally stated by state traces. State traces of transitions systems impose an operational reasoning about the behavior of a system. However this operational behavior can be hidden by using temporal logic to specify safety and liveness properties. Semantics of temporal formulas is given by state-traces of transition systems. A proof system is a way to derive properties from other proved properties without operational reasoning, only by symbolic calculations. Soundness and completeness of the logic are established by proofs relating logical formulas with assertions about state traces. So at this point, we come back to operational reasoning about the transition systems.

In this thesis we propose a more abstract possibility to define the semantics of action systems founded on fixpoints of set (or predicates) transformers. These fixpoints denote the largest subset of states where the iteration of events terminates under the considered fairness assumptions. Our approach is not a matter of predilection or habits, it allows us to make the notion of *reachability* under fairness assumptions equivalent to the one of *termination* of iteration of events. This allows us to prove soundness and completeness of rules proposed to derive *leads-to* properties and it gives the foundations to prove soundness of the rules for the preservation of liveness under refinement. This is the second main contribution of this thesis, in the line of works on denotational semantics as developed in the 60's. We think that it is important to give a common basis to different branches of computer Science, in order to unify the foundation and understanding of our discipline.

This thesis is divided in seven chapters. In Chapter 2 we give an overview of classical formalisms used to specify action systems: UNITY, TLA, Back's action system and event B. In order to better understand similarities and differences among these formalisms, we treat an example of specification and refinement of an abstract producer-consumer system in all of these formalisms.

Chapter 3 presents our approach to the integration of a UNITY-like logic in the specification and proof of liveness properties under minimal progress and weak fairness assumptions. We present proof rules for both the verification and the preservation under refinement of liveness properties. The application of these rules is illustrated in several examples modeling different algorithms of mutual exclusion.

Chapter 4 introduces the semantics of our temporal formulas in terms of fixpoints of set transformers. In this chapter, we define set transformers modeling the iteration of events under minimal progress and weak fairness assumptions. These set transformers allow the definition of a *termination* relation which is used to interpret *leads-to* properties. On another hand, liveness properties holding in a system are denoted by another relation on subsets of states named *reachability*. Soundness and relative completeness of the rules for *leads-to* are proved by showing the equality between these relations.

In Chapter 5 we prove soundness of the rules presented in Chapter 3 to guarantee the preservation of liveness properties in the refinement of event B systems. In Chapter 6 we develop a larger example of mutual exclusion algorithms, under a weak fairness assumption, the refinements of which are intended to be implemented in a centralized or distributed environment.

Finally, in Chapter 7 we draw some conclusions of our work, compare with related works and analyze future work.

Chapter 2

Action and Event Systems

Résumé

Dans ce chapitre nous présentons quatre formalismes qui ont influencé nos recherches : UNITY*,* TLA*, systèmes d'actions et* B *événementiel. Afin de mieux les comparer, nous traitons un exemple simple d'un système producteur-consomateur dans ces formalismes. Ensuite nous synthétisons quelques travaux qui portent sur l'équité de systèmes. Ces travaux traitent des différents aspects tels que la vérification et le développement de programmes et de la sémantique ou mise en œuvre de parallélisme. Finalement, nous faisons une analyse critique de ces travaux et nous proposons notre thèse.*

2.1 UNITY

UNITY [21] is proposed as a theory of parallel programming. It is made up of a computational model and a programming logic. The approach proposes the development of programs in two main steps: first, the solution to a given problem is presented by a UNITY program, which is independent of the architecture and the programming language where the program will be implemented. The second step maps the UNITY program to a particular architecture, which can be a shared memory or message transfer machine. The first step is performed in a rigorous way, by a programming logic. The second step is stated by informal rules indicating how to map the statements of UNITY programs to different architectures. The following subsections give a summary of the derivation of UNITY programs.

2.1.1 UNITY Programs

Mainly, a UNITY program is made up of a *declare* section, where the variables of a program are declared, an *initially* section where initial values are given to the variables and an *assign* section, containing the statements of the program.

The *assign* section is made up of a fix collection of statements s_i separated by the choice operator $[]$:

$$\langle s_1 \; [] \; s_2 \; [] \; \ldots s_n \rangle$$

each s_i can be either an assignment statement or a list of quantified statements.

Statements are executed under an unconditional fairness assumption: any statement is supposed to be infinitely often executed. Termination of programs is considered when they reach a fixpoint: a state where execution of any statement does not change the value of variables.

A statement can be single assignment:

$$\langle \textit{var} \rangle := \langle \textit{expression} \rangle$$

or multiple assignment:

$$\langle \textit{var-list} \rangle := \langle \textit{expression-list} \rangle$$

Assignment statements can be conditioned by boolean expressions:

$$\begin{array}{lcl} \langle \textit{var-list} \rangle & := & \langle \textit{exp-list}_1 \rangle \textbf{ if } \langle \textit{bool-exp}_1 \rangle \sim \\ & & \vdots \\ & & \langle \textit{exp-list}_m \rangle \textbf{ if } \langle \textit{bool-exp}_m \rangle \sim \end{array}$$

An alternative way to write multiple assignments is by the use of the parallel $\|$ operator. In this way, the statement:

$$x_1, \ldots, x_n := e_1 \ldots e_n$$

can be written as:

$$x_1 := e_1 \parallel x_2 := e_2 \parallel \ldots x_n := e_n$$

Quantified statements allow writing list of statements with a regular pattern. They take the general form:

$$\langle \mathsf{op}\ \textit{var-list} \ : \ \textit{range} \ : \ \textit{statement-list} \rangle$$

where op denotes the parallel ($\|$) or choice ($[]$) operator and *range* is a predicate describing the range of the variables. This construction denotes a list of statements, separated by the operator $[]$ or $\|$, and each statement corresponds to the instantiation of *statement-list* to values of the *var-list* in the range *range*.

Example 2.1
The following examples show some assignment statements:

- x takes the value of 1 if p holds or 0 if p does not hold:

  $$x := 1 \textbf{ if } p \ \sim \ 0 \textbf{ if } \neg p$$

- the values of x and y are exchanged:

  $$x := y \parallel y := x$$

- The integer array $A[1..n]$ is initialized to zero:

  $$\langle \| \ i \ : \ 0 \leq i \leq n \ : A[i] := 0 \rangle$$

- A program to sort the integer array $A[1..n]$:

  program *sort*
  assign
  $\langle [] \ i \ : 0 \leq i \leq n \ : A[i], A[i+1] := A[i+1], A[i]$ **if** $A[i] > A[i+1] \rangle$
  end

□

2.1.2 Logic Programming

The logic programming of UNITY is founded around three relations on state predicates: *unless*, *ensures* and *leads-to* [1]. The *unless* relation is used to specify safety properties; liveness properties are specified by *ensures* and *leads-to* relations. These relations are defined by Hoare triplets, universally or existentially quantified over the statements of UNITY programs.

In [21] several theorems about the fundamental relations are presented. These theorems give a formal framework to reason about properties of UNITY programs and they play a fundamental role in the correctness of refinement of UNITY programs.

unless Relation

A property $P \, unless \, Q$ holds in a UNITY program F if at some point in the computation, predicate P becomes *true*, then either Q never holds and P holds forever from this point on, or Q holds eventually and P continues to hold until Q holds. Formally, the *unless* relation is defined by the following inference rule:

$$\frac{\forall s.(s \, \mathbf{in} \, F \Rightarrow \{P \wedge \neg Q\} \ s \ \{P \vee Q\})}{P \, unless \, Q}$$

Example 2.2
Some examples of safety properties:

- The value of x never decreases:

  $$\forall k \cdot (k \in \mathbb{N} \Rightarrow x = k \, unless \, x > k)$$

- Any process p is in state *Idle* until it goes to state *Waiting*:

  $$\forall p \cdot (p \in Proc \Rightarrow p \in Idle \, unless \, p \in Waiting)$$

□

stable

A special case in the definition of *unless* is the case when the predicate Q is equivalent to *false*. It allows the definition of *stable* predicates. Formally, the definition of stable predicates is given by the following inference rule:

$$stable \ P \equiv P \, unless \, false$$

Therefore, P is stable if execution of any statement in a state where P holds terminates into a state where P continues to hold.

[1] Traditionally, the *leads-to* relation in UNITY is denoted by the symbol $\mapsto$. In order to prevent any confusion with the B notation, the *leads-to* relation is denoted by the symbol $\rightsquigarrow$.

invariant
If a predicate P is stable, and it is implied by the initial conditions (IC) of a UNITY program, then it is an *invariant* predicate. That is, an invariant predicate is defined as follows:

$$\textit{invariant}\ P \equiv (IC \Rightarrow P) \wedge \textit{stable}\ P$$

Invariant predicates allow the application of the *substitution axiom*, which states that any invariant predicate in a property can be replaced by the predicate *true* and vice-versa.

ensures Relation

ensures properties allows the specification of basic liveness properties. A property P *ensures* Q holds in a program F, if P *unless* Q holds in F, and moreover there is a helpful statement in F able to establish Q when it is executed in a state where $P \wedge \neg Q$ holds. Formally, this relation is defined by the following rule:

$$\frac{(P\ \textit{unless}\ Q) \wedge \exists s \cdot (s\ \textbf{in}\ F \wedge \{P \wedge \neg Q\}s\{Q\})}{P\ \textit{ensures}\ Q}$$

In the definition of *ensures* properties, the unconditional fairness assumption plays a fundamental role. In fact, it is the only guarantee to expect the helpful statement will be eventually executed.

leads-to Relation

General liveness properties are specified by the *leads-to* relation. If $P \leadsto Q$ holds in a program, then when P becomes *true*, eventually Q becomes *true*. Formally, the *leads-to* relation is defined as the strongest relation satisfying the following rules:

- $P\ \textit{ensures}\ Q \vdash P \leadsto Q$
- $(P \leadsto R) \wedge (R \leadsto Q) \vdash P \leadsto Q$
- For any set W, $\forall m \cdot (m \in W \Rightarrow P(m) \leadsto Q) \vdash \exists m \cdot (m \in W \wedge P(m)) \leadsto Q$

Unlike $P\ \textit{ensures}\ Q$, a property $P \leadsto Q$ cannot assert that P will remain *true* as long as Q is *false*.

Example 2.3
Two examples of *leads-to* properties:

- A process in state waiting eventually becomes active in the critical section:

  $$\forall p \cdot (p \in \mathit{Proc} \Rightarrow p \in \mathit{Waiting} \leadsto p \in \mathit{Active})$$

- In program *sort*, from example 2.2, $A[n]$ eventually gets the maximum value of the array:

  $$\mathit{true} \leadsto A[n] = \mathsf{max}(\mathsf{ran}(A))$$

□

Program Termination

From the unconditional fairness assumption, statements in UNITY programs are infinitely often executed. However, termination of programs can be considered when they reach a state which is stable under the execution of every statement. This state is named a *fixpoint* and it is denoted by a predicate FP. The fixpoint is obtained by replacing the assignment statement symbol := by the equality symbol = in every statement of the program and taking the conjunction over all such predicates.

Example 2.4

- The fixpoint of program *sort* in example 2.2 is:

$$FP \equiv \forall i \cdot (1 \leq i \leq n \Rightarrow A[i] \leq A[i-1])$$

- Termination of program *sort*:

$$true \leadsto FP$$

□

2.1.3 Program Development

UNITY programs are derived from its specification which is made up of safety and liveness properties. Development process proceeds by stepwise refinements. *unless* and *leads-to* relations are used to specify properties of an abstract program. The abstract specification is then gradually refined to concrete properties which are used to derive a program. Statements of the program are often suggested by *ensures* properties which appear in the last refinement steps.

In each refinement step, abstract properties are replaced by concrete ones. The refinement step can be data or algorithmic refinement, and concrete properties specify the behavior of the program in the refined state.

Example 2.5
Let P and C be two variables of type $\mathbb{P}(D)$, where D is any data set. A possible abstract specification considers the liveness property:

$$\forall d \cdot (d \in D \Rightarrow d \in P \leadsto d \in C)$$

which states that any produced data will be eventually consumed.

A common way to synchronize the producer and consumer, is to introduce a *buffer* which is written by the producer and read by the consumer. The buffer is introduced in a refinement and it is modeled by a variable *buf* of type $\mathbb{P}(D)$. Therefore, the abstract liveness property can disappear in this refinement, and two new properties are introduced:

$$d \in P \leadsto d \in buf$$
$$d \in buf \leadsto d \in C$$

where d is universally quantified over D. Correctness of this refinement is simply proved by the transitive property of the *leads-to* relation.

If P, C and buf are implemented by injective sequences SP, SC and SB respectively, and $P = \mathsf{ran}(SP) \wedge C = \mathsf{ran}(SC) \wedge buf = \mathsf{ran}(SB)$, the previous specification can be replaced by two basic properties:

$$d \in \mathsf{ran}(SP) \wedge SP^{-1}(d) = N \textit{ ensures } d \in \mathsf{ran}(SP) \wedge SP^{-1}(d) < N \vee d \in \mathsf{ran}(SB) \quad (2.1)$$

$$d \in \mathsf{ran}(SB) \wedge SB^{-1}(d) = N \textit{ ensures } d \in \mathsf{ran}(SB) \wedge SB^{-1}(d) < N \vee d \in \mathsf{ran}(SC) \quad (2.2)$$

where d and N are universally quantified over D and $\mathbb{N}$ respectively. In order to prove the correctness of this refinement, the induction theorem for *leads-to* properties is used:

$$P \wedge V = n \rightsquigarrow P \wedge V < n \vee Q \vdash P \rightsquigarrow Q$$

where V is a variant function over a well founded set, and n is universally quantified over that set. Taking SP^{-1} as a variant function, $P \equiv d \in \mathsf{ran}(SP)$ and $Q \equiv d \in \mathsf{ran}(SB)$, the induction theorem can be applied to the *leads-to* property derived from (2.1), which gives as result: $d \in \mathsf{ran}(SP) \rightsquigarrow d \in \mathsf{ran}(SB)$. In a similar way, from (2.2) can be derived $d \in \mathsf{ran}(SB) \rightsquigarrow d \in \mathsf{ran}(SC)$. Finally, the correctness of the refinement follows from the equivalence between the abstract and concrete data.

Properties (2.1) and (2.2) suggest two helpful statements to access the main data structures. The following statements can be part of a UNITY program which implements the producer-consumer system:

$$\langle SB, SP := SB \leftarrow \mathsf{first}(SP), \mathsf{tail}(SP) \textbf{ if } SP \neq [\,] \;[]$$
$$SC, SB := SC \leftarrow \mathsf{first}(SB), \mathsf{tail}(SB) \textbf{ if } SB \neq [\,] \rangle$$

□

2.1.4 New Unity

In [48, 49] the logic programming of UNITY has been revisited. Fundamental relations *unless* is replaced by a new relation named *constrains* (*co*) and the *ensures* relation is defined by the notion of *transient* predicate. The new logic is intended to be applied not only to UNITY programs, but to any action system in general. The only imposed restriction is that there is an *skip* statement in the system that may be applied in any program state.

constrains relation becomes the fundamental relation to specify safety properties. For any predicate P and Q, $P \textit{ co } Q$ specifies that whenever P holds, Q holds after execution of any statement. Formally, the definition of the relation *constrains* for any action system A is:

$$P \textit{ co } Q \;\widehat{=}\; \forall s \cdot (s \textbf{ in } A \wedge P \Rightarrow wlp(s, Q))$$

From this definition, from $P \textit{ co } Q$ follows $P \Rightarrow Q$, because *skip* is in A. Moreover, if P holds, it continues to hold until $\neg P \wedge Q$ holds.

The *unless* relation can be defined by the *constrains* relation:

$$P \textit{ unless } Q \equiv P \wedge \neg Q \textit{ co } P \vee Q$$

Therefore, a *stable* predicate becomes:

$$\textit{stable } P \equiv P \textit{ co } P$$

In [48] another series of theorems for the *constrains* relation can be found.

transient predicates are predicates which are guaranteed to be falsified by execution of a single atomic action. The formal definition of *transient* predicates depends on the fairness assumption taken in the action system. In [49], *transient* predicates are defined for systems with minimal progress and weak fairness assumptions.

For minimal progress, definition of *transient* predicates consider action systems which take the form of guarded commands:

$$\langle [\!] \ i :: g_i \rightarrow s_i \rangle$$

Considering that execution of particular actions cannot be guaranteed under this assumption, a predicate is *transient* if it is falsified by any action in the system. However, in order to be executed, at least one action need to be enabled. The formal definition of *transient* predicates under minimal progress assumption is given by the following rule:

$$(P \Rightarrow \exists i \cdot (g_i)) \wedge \forall i \cdot (P \wedge g_i \Rightarrow wlp(s_i, \neg P)) \vdash transient\, P$$

For weak fairness assumptions, a predicate is *transient* if it is falsified by an action of the system. Formally, for any action system A, the definition is:

$$transient\, P \ \widehat{=} \ \exists s \cdot (s \ \mathbf{in}\ A \wedge P \Rightarrow wlp(s, \neg P))$$

The *ensures* relation can now be defined by *transient* predicates:

$$P \ ensures\, Q \equiv (P \wedge \neg Q \ co\ P \vee Q) \wedge (P \wedge \neg Q \ transient)$$

This definition allows one to apply the previous rules for *leads-to*, and then derive general liveness properties.

2.2 TLA

TLA [41] is a logic to reason about concurrent algorithms. It combines two logics: a logic of actions and a standard linear temporal logic. The logic of actions allows the specification of state changes by a before-after relation between state variables. The same logic is used to specify an algorithm and its properties, simplifying the proof that an implementation satisfies its specification.

2.2.1 The Logic of Actions

The logic considers an infinite set of variables Var and a collection of values Val. Variables have no type, and they can assume any value. Type correctness is a provable property in a specification and it is not a syntactic requirement. Semantics is defined in terms of states, which are functions from Var to Val. The set of states is denoted by St. Predicates P are boolean expressions built from variables and constants, and its meaning $[\![P]\!]$ is a mapping from states to booleans.

Actions are boolean expressions formed from variables, primed variables and constant symbols. Actions represent a relation between old states and new states, where the unprimed variables refer to the old state and the primed variables refer to the new state.

Example 2.6
Some examples of actions:

- This action asserts that the value of y in the old state is one greater than the value of x in the new state:

  $$y = x' + 1$$

- This action asserts that a belongs to the set *Data*, that the size of variable *buf* in the old state is lower than N and that the new value of the buffer *buf* is equal to the concatenation of its old value with the list $\langle a \rangle$:

  $$a \in \mathit{Data} \wedge \mathsf{size}(\mathit{buf}) < N \wedge \mathit{buf}' = \mathit{buf} \circ \langle a \rangle$$

□

Formally, the meaning $[\![\mathcal{A}]\!]$ of an action $\mathcal{A}$ is a relation between old and new states. Therefore, for any states s and t, $s[\![\mathcal{A}]\!]t$ holds if and only if execution of the operation denoted by action $\mathcal{A}$ in state s, terminates in a state t; in this case, the pair of states s and t is called an "$\mathcal{A}$ step". Predicates can be seen as actions with no primed variables. Therefore, for any predicate P, and states s and t, the boolean $s[\![P]\!]t$ is equivalent to $s[\![P]\!]$.

Each action $\mathcal{A}$ has associated a predicate $\mathit{Enabled}\,\mathcal{A}$ which holds in state s if and only if it is possible to take an $\mathcal{A}$ step starting in s. That is:

$$s[\![\mathit{Enabled}\,\mathcal{A}]\!] \mathrel{\widehat{=}} \exists t \in \mathsf{St} : s[\![\mathcal{A}]\!]t$$

2.2.2 Simple Temporal Logic

Temporal formulas are made up of elementary formulas, boolean operators and the *always* operator □. TLA is defined as a special case of the simple temporal logic, by specifying its elementary formulas.

Semantics of temporal formulas is defined in terms of behaviors, which are infinite sequences of states. Thus, the meaning $[\![F]\!]$ of a temporal formula F, is a mapping from behaviors to booleans. Since boolean operators can be defined in terms of conjunction and negation of formulas, it suffices to give the following definitions for any behavior $\sigma = \langle s_0, s_1, s_2, \ldots \rangle$ and states s_i:

$$\begin{aligned} \sigma\,[\![F \wedge G]\!] &\mathrel{\widehat{=}} \sigma\,[\![F]\!] \wedge \sigma[\![G]\!] \\ \sigma\,[\![\neg F]\!] &\mathrel{\widehat{=}} \neg\,\sigma\,[\![F]\!] \\ \sigma\,[\![\Box F]\!] &\mathrel{\widehat{=}} \forall n \in \mathbb{N} : \langle s_n, s_{n+1}, s_{n+2}, \ldots \rangle [\![F]\!] \end{aligned}$$

From the semantics of the *always* operator, it follows that $\Box\, F$ asserts that F is always *true*.

From the *always* operator, the *eventually* operator $\Diamond$ is defined, for any temporal formula F:

$$\Diamond F \mathrel{\widehat{=}} \neg\Box\neg F$$

Thus, from the meaning of the *always* operator, the semantics of the *eventually* operator is:

$$\langle s_0, s_1, s_2, \ldots \rangle [\![\Diamond F]\!] \equiv \exists n \in \mathbb{N} : \langle s_n, s_{n+1}, s_{n+2}, \ldots \rangle [\![F]\!]$$

For temporal formulas F and G, the *leads-to* operator is defined as follows:

$$F \rightsquigarrow G \mathrel{\widehat{=}} \Box(F \Rightarrow \Diamond G)$$

It asserts that any time F is *true*, G is *true* then or at some later time.

2.2.3 The Raw Temporal Logic

The raw temporal logic is obtained from the simple temporal logic by letting the elementary temporal formulas be actions. An action $\mathcal{A}$ holds in a behavior if and only if the first pair of states is an $\mathcal{A}$ step:

$$\langle s_0, s_1, s_2, \ldots \rangle [\![\mathcal{A}]\!] \mathrel{\widehat{=}} s_0 [\![\mathcal{A}]\!] s_1$$

From the previous definitions follows that a behavior satisfies $\Box\mathcal{A}$ if and only if every step of the behavior is an $\mathcal{A}$ step:

$$\langle s_0, s_1, s_2, \ldots \rangle [\![\Box\mathcal{A}]\!] \equiv \forall n \in \mathbb{N} : s_n [\![\mathcal{A}]\!] s_{n+1}$$

As indicated above, a predicate P can be seen as an action without primed variables. Therefore, a behavior satisfies P if and only if the first state of the behavior satisfies P, and it satisfies $\Box P$ if and only if all sates of the behavior satisfy P:

$$\begin{array}{rcl} \langle s_0, s_1, s_2, \ldots \rangle [\![P]\!] & \equiv & s_0 [\![P]\!] \\ \langle s_0, s_1, s_2, \ldots \rangle [\![\Box P]\!] & \equiv & \forall n \in \mathbb{N} : s_n [\![P]\!] \end{array}$$

Example 2.7
In the following example an abstract producer-consumer system is specified by a formula of the raw temporal logic. The specification uses a constant set (rigid variable) D and variables P and C. Predicate *Init* specifies the initial values of these variables:

$$\mathit{Init} \quad \widehat{=} \quad P = D \wedge C = \varnothing$$

The system is made out of two actions: $pc(d)$ which models production and consumption of item d and *env* which models the environment. Action *pc* transfers data from P to C and *env* resets P and C to their original values:

$$\begin{array}{rcl} pc(d) & \widehat{=} & d \in P \wedge P' = P \setminus \{d\} \wedge C' = C \cup \{d\} \\ \mathit{env} & \widehat{=} & P = \varnothing \wedge P' = D \wedge C' = \varnothing \end{array}$$

The following raw temporal formula Φ_0 holds in a behavior $\langle s_0, s_1, s_2, \ldots \rangle$ if and only if *Init* holds in s_0 and any step of the behavior (s_i, s_{i+1}), for any i in $\mathbb{N}$, is a step where *pc* or *env* holds:

$$\Phi_0 \quad \widehat{=} \quad \mathit{Init} \wedge \Box(\exists d \in D : pc(d) \vee \mathit{env})$$

□

2.2.4 Including Stuttering Steps

A stuttering step is a pair of consecutive states in a behavior where (state) variables do not change their values. TLA is defined as a subset of the raw temporal logic where formulas are invariant under *stuttering* steps. It means that adding or removing stuttering steps from a behavior does not affect whether the behavior satisfies the temporal formulas.

In order to ensure that a formula is invariant under stuttering steps, the *square* $\mathcal{A}$ *sub* f notation is introduced:

$$[\mathcal{A}]_f \mathrel{\widehat{=}} \mathcal{A} \vee f = f'$$

where $\mathcal{A}$ is an action and f a term (state function). In general f denotes a tuple $\langle v_1, v_2, \ldots v_n \rangle$ of variables v_i, f' represents the same tuple with primed variables and the equality $f = f'$ is: $v_1 = v_1' \wedge v_2 = v_2' \wedge \ldots v_n = v_n'$.

In this way, the elementary formulas of TLA are predicates and formulas of the form $\Box[\mathcal{A}]_f$, where $\mathcal{A}$ is an action and f a state function.

Example 2.8
The specification Φ_0 of the producer-consumer system in the example 2.7 can be rewritten as the following TLA formula:

$$\Phi_1 \quad \widehat{=} \quad \mathit{Init} \wedge \Box[\exists d \in D : \mathit{pc}(d) \vee \mathit{env}]_{\langle P,C \rangle}$$

Behaviors satisfying Φ_0 fulfill the requirements of Φ_1. Moreover, the TLA specification allows stuttering steps, where the variables P and C do not change their values. □

2.2.5 Considering Fairness Conditions

TLA formulas specifying *safety properties* as $\mathit{Init} \wedge \Box[\mathcal{A}]_f$, where *Init* is a predicate indicating the initial values of the state function f and $\mathcal{A}$ an action or disjunction of actions, can be satisfied by behaviors made out of solely stuttering steps. These behaviors do not allow the dynamic evolution of the specified system and they must be disallowed. In order to restrict stuttering steps in a behavior, TLA formulas must be strengthened by *fairness assumptions* which allow the expression of *liveness properties*. TLA considers *weak* and *strong* fairness assumptions.

In order to specify fairness assumptions, the *angle* $\mathcal{A}$ *sub* f notation is introduced:

$$\langle \mathcal{A} \rangle_f \widehat{=} \mathcal{A} \wedge (f \neq f')$$

This notation species the execution of the operation modeled by the action $\mathcal{A}$, which modifies the value of the state function f. A simple calculation can show that the following equivalence holds:

$$\Diamond \langle \mathcal{A} \rangle_f \equiv \neg \Box [\neg \mathcal{A}]_f$$

which proofs that $\langle \mathcal{A} \rangle_f$ is a TLA formula.

A *weak fairness* assumption asserts that an operation, modeled by an action $\mathcal{A}$, is infinitely often executed if it is continuously enabled. Formally, the weak fair execution of the operation modeled by $\mathcal{A}$ ($\mathrm{WF}_f(\mathcal{A})$) is defined as follows:

$$\mathrm{WF}_f(\mathcal{A}) \quad \widehat{=} \quad \Box(\Box \mathit{Enabled} \langle \mathcal{A} \rangle_f \Rightarrow \Diamond \langle \mathcal{A} \rangle_f)$$

A *strong fairness* assumption asserts that an operation, modeled by an action $\mathcal{A}$, is infinitely often executed if it is infinitely often enabled. Formally, the strong fair execution of the operation modeled by $\mathcal{A}$ ($\mathrm{SF}_f(\mathcal{A})$) is defined as follows:

$$\mathrm{SF}_f(\mathcal{A}) \quad \widehat{=} \quad \Box\Diamond \mathit{Enabled} \langle \mathcal{A} \rangle_f \Rightarrow \Box\Diamond \langle \mathcal{A} \rangle_f$$

Example 2.9
In order to limit the stuttering steps in the behaviors satisfying specification Φ_1 of example

2.8, the specification of the producer-consumer system can be strengthened by a weak fair condition on actions $pc(d)$ and env:

$$\Phi_2 \;\widehat{=}\; Init \wedge \Box[\exists d \in D : pc(d) \vee env]_{\langle P,C\rangle} \wedge \forall d \in D : \mathrm{WF}_{\langle P,C\rangle}(pc(d)) \wedge \mathrm{WF}_{\langle P,C\rangle}(env)$$

Behaviors satisfying Φ_2 contain infinitely many steps where a data d is transfered from P to C. □

Sometimes it is possible to specify a conjunction of weak fairness conditions $\mathrm{WF}_v(\mathcal{A}_0) \wedge \mathrm{WF}_v(\mathcal{A}_1) \wedge \ldots \mathrm{WF}_v(\mathcal{A}_n)$, by an equivalent condition on the disjunction of actions $\mathrm{WF}_v(\mathcal{A}_0 \vee \mathcal{A}_1 \vee \ldots \mathcal{A}_1)$. The equivalence depends on the WF Conjunction Rule, stated as follows:

> **WF Conjunction Rule** If $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are actions such that, for any distinct i and j, whenever $\langle \mathcal{A}_i \rangle_v$ is enabled, $\langle \mathcal{A}_j \rangle_v$ cannot become enabled unless an $\langle \mathcal{A}_i \rangle_v$ step occurs, then $\mathrm{WF}_v(\mathcal{A}_0) \wedge \mathrm{WF}_v(\mathcal{A}_1) \wedge \ldots \mathrm{WF}_v(\mathcal{A}_n)$ is equivalent to $\mathrm{WF}_v(\mathcal{A}_0 \vee \mathcal{A}_1 \vee \ldots \mathcal{A}_n)$.

An analogous conjunction rule exists for strong fairness.

Example 2.10
The weak fairness conditions on actions $pc(d)$ and env, in the formula Φ_2 of example 2.9, can be rewritten in an equivalent way, as the following formula shows:

$$\Phi_2 \;\widehat{=}\; Init \wedge \Box[\exists d \in D : pc(d) \vee env]_{\langle P,C\rangle} \wedge \forall d \in D : \mathrm{WF}_{\langle P,C\rangle}(pc(d) \vee env)$$

The equivalence follows from the WF conjunction rule and two facts:

- For any d in D, the action $pc(d)$ is not enabled when the action env is enabled and
- For any d in D, the action env is not enabled when action $pc(d)$ is enabled.

□

2.2.6 Hiding Variables

Apart from traditional quantification over constants (rigid variables), TLA allows existential quantification over (state) variables (flexible variables). This kind of quantification permits hiding of variables in temporal formulas.

Informally, a temporal formula $\exists x : F$, where x is a state variable and F a temporal formula where x is free, asserts that any state of any behavior satisfying the quantified formula, contains a value of x, such that F holds. Therefore $\exists x : F$ asserts the existence of infinitely many values of x satisfying F.

Formally, the meaning of the quantifier $\exists$ is defined as comes next:

$$\sigma[\![\exists x : F]\!] \mathrel{\widehat{=}} \exists \rho, \tau : \mathsf{St}^\infty : (\natural\sigma = \natural\rho) \wedge (\rho =_x \tau) \wedge \tau[\![F]\!]$$

where St^∞ is the set of behaviors, the equality $\rho =_x \tau$ states that any states[2] τ_i and σ_i assign the same value to all other variables except x, and $\natural$ is a function over behaviors where $\natural\sigma$, for any behavior σ, is the behavior obtained from σ by removing all stuttering steps except the final ones. The quantifier $\exists$ obeys the ordinary laws of existential quantification, in particular it distributes over disjunction of temporal formulas: $(\exists x : F \vee G) \equiv (\exists x : F) \vee (\exists x : G)$.

[2]For any behavior $\langle s_0, s_1, \ldots\rangle$, $\langle s_0, s_1, \ldots\rangle_i$ denotes s_i.

Example 2.11
From the example 2.9, the variable C can be hidden from the formula Φ_2:

$$\Phi_3 \;\widehat{=}\; \exists C : \mathit{Init} \wedge \Box[\exists d \in D : \mathit{pc}(d) \vee \mathit{env}]_{\langle P,C\rangle} \wedge \forall d \in D : \mathrm{WF}_{\langle P,C\rangle}(\mathit{pc}(d)) \wedge \mathrm{WF}_{\langle P,C\rangle}(\mathit{env})$$

In Φ_3 it does not matter the value of the variable C. The formula only asserts the existence of certain values that C can assume for which Φ_2 holds □

2.2.7 Refinement of Specifications

Specifications of systems in TLA are written as a conjunction of tree formulas:

$$\Phi \;\widehat{=}\; I \wedge \Box[\mathcal{N}]_f \wedge F$$

where :

- I is a predicate specifying the initial values of variables,
- $\mathcal{N}$ is the disjunction of actions in the system; generally $\mathcal{N}$ is known as the *next state* relation,
- f is a tuple made up of all variables in the system and
- F is a conjunction of formulas of the form $\mathrm{WF}(\mathcal{A})$ and/or $\mathrm{SF}(\mathcal{A})$ and $\mathcal{A}$ is an action or disjunction of actions.

In general, some variables $x_1, \ldots, x_m$ of Φ are hidden: $\exists x_1, \ldots, x_m : \Phi$. Development of systems specified in this way is made by stepwise refinements. Refinements are again temporal formulas, analogous to specifications, where refined actions transform the concrete state variables.

Considering the formula $\exists y_1, \ldots, y_n : \Psi$ as a refinement of the system specified by $\exists x_1, \ldots, x_m : \Phi$, correctness of such a refinement is proved by logical implication:

$$\exists x_1, \ldots, x_m : \Phi \Rightarrow \exists y_1, \ldots, y_n : \Psi$$

In order to prove this theorem, a series of state functions $\overline{y_1}, \ldots, \overline{y_n}$ must be defined in terms of the variables that occur in Ψ and prove $\Psi \Rightarrow \overline{\Phi}$, where $\overline{\Phi}$ denotes the formula Φ where each occurrence of variable y_i is replaced by $\overline{y_i}$, for all i in $1..n$. The set of state functions is called a *refinement mapping*, and each $\overline{y_i}$ denotes the concrete variable with which Ψ implements y_i.

TLA disposes of a set of proof rules to prove TLA theorems. As an example the rules TLA2 and WF2 are presented:

TLA2

$$\frac{P \wedge [\mathcal{A}]_f \Rightarrow Q \wedge [\mathcal{B}]_g}{\Box P \wedge \Box[\mathcal{A}]_f \Rightarrow \Box Q \wedge \Box[\mathcal{B}]_g}$$

WF2

$$\frac{\begin{array}{l}\langle \mathcal{N} \wedge \mathcal{B}\rangle_f \Rightarrow \langle \overline{\mathcal{M}}\rangle_{\overline{g}} \\ P \wedge P' \wedge \langle \mathcal{N} \wedge \mathcal{A}\rangle_f \wedge \overline{\mathit{Enabled}\langle \mathcal{M}\rangle_g} \Rightarrow \mathcal{B} \\ P \wedge \overline{\mathit{Enabled}\langle \mathcal{M}\rangle_g} \Rightarrow \mathit{Enabled}\langle \mathcal{A}\rangle_f \\ \Box[\mathcal{N} \wedge \neg\mathcal{B}]_f \wedge \mathrm{WF}_f(\mathcal{A}) \wedge \Box F \wedge \\ \qquad \Diamond\Box\overline{\mathit{Enabled}\langle \mathcal{M}\rangle_g} \Rightarrow \Diamond\Box F\end{array}}{\Box[\mathcal{N}]_f \wedge \mathrm{WF}_f(\mathcal{A}) \wedge \Box F \Rightarrow \overline{\mathrm{WF}_g(\mathcal{M})}}$$

TLA2 is useful to prove *step simulations* when $\mathcal{A}$ and $\mathcal{B}$ denote the next state relation of the abstraction and the refinement of a system respectively. WF2 is used to deduce the weak fairness condition of an action in the refinement, from the weak fairness condition of another action in the abstraction. The following example shows how to use these rules.

Example 2.12
In this example the producer-consumer system, introduced in the example 2.7 and specified by the formula Φ_3 of example 2.11 is refined. In this refinement, the producer and the consumer communicate by an infinite shared buffer. The rigid variable D is conserved as well as the variable P. Two new variables are introduced C_c and buf. Initially, the values of these variables are specified by following formula:

$$Init_c \;\widehat{=}\; P = D \wedge C_c = \varnothing \wedge buf = \langle\rangle$$

The behavior of the system in the refinement is modeled by three actions: $prod(d)$, $cons$ and env. Action $prod(d)$ models the production of item d by a transfer of d from P to buf. Action $cons$ models consumption of an item by transferring it from buf to C_c. Finally, action env, as in the abstract specification, models the environment which restart a new cycle of production-consumption of items, by reseting the variables to their initial values. Formally, the actions are:

$$\begin{aligned} prod(d) &\;\widehat{=}\; d \in P \wedge P' = P - \{d\} \wedge buf' = buf \circ \langle d\rangle \wedge C_c' = C_c \\ cons &\;\widehat{=}\; buf \neq \langle\rangle \wedge C_c' = C_c \cup \{\mathsf{first}(buf)\} \wedge buf' = \mathsf{tail}(buf) \wedge P' = P \\ env_c &\;\widehat{=}\; P = \varnothing \wedge buf = \langle\rangle \wedge P' = D \wedge C_c' = \varnothing \wedge buf' = buf \end{aligned}$$

The refinement is specified by the following formula:

$$\Psi \;\widehat{=}\; Init_c \wedge \Box[\mathcal{N}_c]_w \wedge \forall d \in D : \mathrm{WF}_w(prod(d)) \wedge \mathrm{WF}_w(cons) \wedge \mathrm{WF}_w(env_c)$$

where

$$\begin{aligned} \mathcal{N}_c &\;\widehat{=}\; \exists d \in D : prod(d) \vee cons \vee env \\ w &\;\widehat{=}\; \langle P, C_c, buf\rangle \end{aligned}$$

In order to prove that Ψ implements the abstract producer-consumer system, specified by Φ_3, the following implication must be proved:

$$\Psi \Rightarrow \exists C : \Phi_2$$

where Φ_2 is defined in the example 2.9 on page 16. The proof requires the definitions of the following refinement mapping:

$$\overline{C} = C_c \cup \mathsf{ran}(buf)$$

which states that the set C, used in the abstraction, is implemented by the union of the concrete set C_c and the buffer buf. Therefore, the correctness of the refinement follows from:

$$\Psi \Rightarrow \overline{\Phi_2}$$

where $\overline{\Phi_2}$ is Φ_2 with each occurrence of C replaced by $\overline{C}$:

$$\overline{\Phi_2} \;\widehat{=}\; \overline{Init} \wedge \Box[\exists d \in D : \overline{pc(d)} \vee \overline{env}]_{\langle P,\overline{C}\rangle} \wedge \forall d \in D : \overline{\mathrm{WF}_{\langle P,C\rangle}(pc(d))} \wedge \overline{\mathrm{WF}_{\langle P,C\rangle}(env)}$$

$\overline{Init}$, $\overline{pc(d)}$ and $\overline{env}$ are $Init$, $pc(d)$ and env respectively, with each occurrence of C replaced by $\overline{C}$:

$$\begin{array}{rcl} \overline{Init} & \widehat{=} & P = D \wedge \overline{C} = \varnothing \\ \overline{pc(d)} & \widehat{=} & d \in P \wedge P' = P \setminus \{d\} \wedge \overline{C}' = \overline{C} \cup \{d\} \\ \overline{env} & \widehat{=} & P = \varnothing \wedge P' = D \wedge \overline{C}' = \varnothing \end{array}$$

The proof follows from the following implications:

- $Init_c \Rightarrow \overline{Init}$
- $\Box[\mathcal{N}_c]_w \Rightarrow \Box[\exists d \in D : \overline{pc(d)} \vee \overline{env}]_{\langle P,\overline{C}\rangle}$ and
- $\forall d \in D : \mathrm{WF}_w(prod(d)) \wedge \mathrm{WF}_w(cons) \wedge \mathrm{WF}_w(env_c) \Rightarrow$
  $\forall d \in D : \overline{\mathrm{WF}_{\langle P,C\rangle}(pc(d))} \wedge \overline{\mathrm{WF}_{\langle P,C\rangle}(env)}$

The first implication follows directly from definitions of $Init_c$, $\overline{C}$ and $\overline{Init}$.

The second implication follows by applying the rule TLA2. Instantiating P and Q to *true*, the premise to that rule requires to prove:

$$[\mathcal{N}_c]_w \Rightarrow [\exists d \in D : \overline{pc(d)} \vee \overline{env}]_{\langle \overline{C}\rangle}$$

This premise follows from the following implication which are proved from definitions:

$$\begin{array}{l} prod(d) \Rightarrow \overline{pc(d)} \\ cons \Rightarrow \langle P', \overline{C}'\rangle = \langle P, \overline{C}\rangle \\ env \Rightarrow \overline{env} \\ w' = w \Rightarrow \langle P', \overline{C}'\rangle = \langle P, \overline{C}\rangle \end{array}$$

The last implication follows by applying twice the rule WF2: one to deduce $\overline{\mathrm{WF}_{\langle P,C\rangle}(pc(d))}$ and another one to prove $\overline{\mathrm{WF}_{\langle P,C\rangle}(env)}$. In this example, only the proof of $\overline{\mathrm{WF}_{\langle P,C\rangle}(pc(d))}$ is outlined. Instantiating WF2 with the following substitutions:

$$\mathcal{N} \leftarrow true \quad \mathcal{B} \leftarrow prod(d) \quad \mathcal{A} \leftarrow prod(d) \quad P \leftarrow true \quad F \leftarrow true \quad \mathcal{M} \leftarrow pc(d) \quad f \leftarrow w \quad g \leftarrow \langle P, C\rangle$$

gives the following rule:

$$\frac{\begin{array}{l} \langle prod(d)\rangle_w \Rightarrow \langle \overline{pc(d)}\rangle_{\langle P,\overline{C}\rangle} \\ \overline{Enabled\langle pc(d)\rangle_{\langle P,C\rangle}} \Rightarrow Enabled\langle prod(d)\rangle_w \end{array}}{\mathrm{WF}_w(prod(d)) \Rightarrow \overline{\mathrm{WF}_{\langle P,C\rangle}(pc(d))}}$$

The premise of this rule is easily derived from definitions, which allows the proof of the goal. □

2.3 The Action System Formalism

Action systems were introduced in [10] as an execution model, intended to replace the conventional process-oriented execution model by an action-oriented model, where atomic actions can be executed whenever they are enabled, and the selection among them is nondeterministic. Subsequent work [13, 15, 12] led to formalizing actions systems in the refinement calculus.

2.3.1 The Refinement Calculus

The refinement calculus is the mathematical foundation of the stepwise refinement method of program construction. The basic domains of this calculus are obtained by pointwise extension of the boolean lattice.

The set of booleans Bool, $\mathsf{Bool} = \{\mathsf{T}, \mathsf{F}\}$, forms a complete lattice under the implication order : $\mathsf{F} \leq \mathsf{T}$, $\mathsf{T} \leq \mathsf{T}$ and $\mathsf{F} \leq \mathsf{F}$. The complement, meet and join of this lattice are $\neg$, $\wedge$ and $\vee$ respectively.

Predicates are defined as functions from the state space Σ to the booleans: $\mathsf{Pred}(\Sigma) \mathrel{\widehat{=}} \Sigma \rightarrow \mathsf{Bool}$. This set forms a complete lattice under the order obtained from logical implication by pointwise extension. That is, for any P and Q in $\mathsf{Pred}(\Sigma)$, $P \leq Q \equiv (\forall \sigma \in \Sigma \cdot P\,\sigma \leq Q\,\sigma)$. Complement, meet and join are defined by pointwise extension too. The bottom in this lattice is *false* and the top *true*.

Semantics of program statements in the calculus are given in terms of *predicates transformers*. Predicate transformers are defined by pointwise extension of predicates. For any state spaces Σ and Γ, the set of predicate transformers $\mathsf{Ptran}(\Sigma, \Gamma)$ is defined as the set of functions from predicates on Γ to predicates on Σ: $\mathsf{Ptran}(\Sigma, \Gamma) \mathrel{\widehat{=}} \mathsf{Pred}(\Gamma) \rightarrow \mathsf{Pred}(\Sigma)$. For any predicate transformer A in $\mathsf{Ptran}(\Sigma, \Gamma)$ and predicate P on Γ, the predicate $A\,P$ on Σ denotes the *weakest precondition* where the statement modeled by A can start execution and terminates into a state satisfying P.

Refinement The *refinement order* for any T and S in $\mathsf{Ptran}(\Sigma, \Gamma)$ is defined by

$$S \leq T \equiv \forall Q \in \mathsf{Pred}(\Gamma) \cdot S\,Q \leq T\,Q$$

This order means that S refines T if and only if S establishes any postcondition that T does. Under this order, predicate transformer form a lattice with bottom abort and top magic. abort maps any predicate to *false* and magic to *true*. Meet models *demonic* (internal) choice and join (external) *angelic* choice. Meet and join are defined by pointwise extension e.g. $(S \wedge T)\,Q \mathrel{\widehat{=}} S\,Q \wedge T\,Q$.

The non miraculous domain or *guard* of a predicate transformer A is defined by

$$gA \mathrel{\widehat{=}} \neg A\,\mathit{false}$$

and the termination domain of A is

$$tA \mathrel{\widehat{=}} A\,\mathit{true}$$

The statement modeled by A is enabled in any state where gA holds, and its termination is guaranteed in any state where tA holds.

Data Refinement Stepwise refinement of statements often leads to changes in the state space of the refined statement. In this case, the refinement involves replacing the abstract variables a with concrete variables c which are more easily implemented. However, both the abstract and concrete states, have a common variable x which is not modified by refinement. In this way the abstract state space becomes $\Psi \times \Sigma_S$ where Ψ is the state space of x and Σ_S the state space of the abstract variable a. In a similar way, the concrete state becomes $\Psi \times \Gamma_T$ where Γ_T is the state space of the concrete variable c. The correctness proof of data

refinement requires to define the abstract relation $R(x, a, c)$ which relates the concrete variable with the abstract and common variables. In this way, the refinement of an abstract predicate transformer S in $\mathsf{Ptran}(\Sigma, \Sigma)$, where $\Sigma = \Psi \times \Sigma_S$, by a concrete predicate transformer T in $\mathsf{Ptran}(\Gamma, \Gamma)$, where $\Gamma = \Psi \times \Gamma_T$, is defined as follows

$$S \leq_R T \mathrel{\widehat{=}} R(x, a, c) \wedge \\ S\,Q(x, a) \leq T\,(\lambda(x, c) \cdot (\exists a' \in \Sigma_S \cdot R(x, a', c) \wedge Q(x, a')))(x, c) \tag{2.3}$$

where Q is universally quantified over $\mathsf{Pred}(\Sigma)$, and the variables x, a and c are universally quantified over their respective state spaces. In this definition it must be remarked that $\lambda(x, c) \cdot (\exists a' \in \Sigma_S \cdot R(x, a', c) \wedge Q(x, a'))$, and $T(\lambda(x, c) \cdot (\exists a' \in \Sigma_S \cdot R(x, a', c) \wedge Q(x, a')))$ are predicates over Γ while $S\,Q$ is a predicate over Σ. Moreover, the order relation in the definition corresponds to the implication order in the boolean lattice.

2.3.2 Action System

Action systems allows the extension of development methods for sequential programs to the design and implementation of parallel and reactive systems. They are able to model terminating, possibly aborting and infinitely repeating system.

An action system $\mathcal{A}$ is a statement in the command language of the refinement calculus taking the form:

$$\mathcal{A} \mathrel{\widehat{=}} |[\mathsf{var}\, x \bullet Q \;;\; \mathsf{do}\, A_1 \,[\!]\, A_2 \,[\!]\, \ldots A_n\, \mathsf{od}]| : z$$

where x and z are tuples of local and global variables respectively, Q is the initialization condition and each A_i is an *action* taking the guarded command form:

$$A_i \mathrel{\widehat{=}} g_i \rightarrow S_i$$

where g_i is the guard of the action and S_i its body. The body of A_i is supposed to be non miraculous ($S_i\, \mathit{false} \equiv \mathit{false}$) and universally conjunctive ($S_i \bigwedge_j q_j \equiv \bigwedge_j S_i\, q_j$ for any subset of predicates q_j).

Local and global variables are assumed to be distinct and they determine the state space of the system. After initialization, the state of the action system satisfies the initial condition. The iteration of actions determines what can happen in the system during an execution. The body of any action can transform the state if it is executed in any state where the guard of the action holds. The iteration of actions terminates when no action is enabled, that is, when the action system arrives into a state where the guard of every action does not hold.

Example 2.13
In this example the abstract producer-consumer system, treated in the previous examples, is specified as an action system. The variables P and C, of type $\mathbb{P}(D)$, are considered as global and local variables respectively. The initial state must satisfy the condition $P = D \wedge C = \varnothing$. The goal of the system is to transfer data from P to C. This is achieved by the action system PC:

$$PC \quad \mathrel{\widehat{=}} \quad |[\mathsf{var}\, C \bullet P = D \wedge C = \varnothing \;;\; \mathsf{do}\, \mathit{pc} \,[\!]\, \mathit{env}\, \mathsf{od}]| : P$$

where the actions *pc* and *env* are [3]:

$$\begin{aligned} pc &\;\widehat{=}\; P \neq \varnothing \rightarrow P, C := P - \{d\}, C \cup \{d\} \cdot d \in P \\ env &\;\widehat{=}\; P = \varnothing \rightarrow P, C := D, \varnothing \end{aligned}$$

□

Action system can be composed by a parallel operator $\|$. Two or more action systems can be composed if their global variables are the same and their local variables are disjoints. Therefore, if the action system $\mathcal{A}_i$ is defined by

$$\mathcal{A}_i \;\widehat{=}\; |[\mathsf{var}\, x_i \bullet p_i \;;\; \mathsf{do}\, A_i \,\mathsf{od}]| : z$$

the parallel composition $\mathcal{A}_1 \parallel \mathcal{A}_2$ is defined as follows

$$\mathcal{A}_1 \parallel \mathcal{A}_2 \;\widehat{=}\; |[\mathsf{var}\, x_1, x_2 \bullet p_1 \wedge p_2 \;;\; \mathsf{do}\, A_1 \,[\!]\, A_2 \,\mathsf{od}]| : z$$

2.3.3 Behavioral Semantics

A behavior or computation σ of an action system $|[\mathsf{var}\, x \bullet p \;;\; \mathsf{do}\, A \,\mathsf{od}]| : z$ is a finite or infinite sequence of states generated by an execution of the action system:

$$\sigma = \langle s_0, s_1, \ldots \rangle$$

Each state s_i in the behavior is a pair of values (a_i, u_i), where a_i denotes the value of the local variables x and u_i the value of the global variables z.

An *infinite computation* σ is an infinite sequence of states $\langle s_0, s_1, \ldots \rangle$ where s_0 satisfies the initial condition p of the action system, and the *next state relation* associated to A holds for any consecutive states s_i and s_{i+1} of σ. A *terminated computation* is a finite sequence where no action is enabled in the last state and an *aborted computation* is a finite sequence where termination of the action system is not guaranteed in the last state.

Traces of global states are obtained from computations by deleting local states and stuttering transitions. Stuttering transitions are steps that do not change global states. When an infinite computation contains only stuttering steps from certain point onwards, the resulting trace is considered to be aborted. Moreover, aborted computations induce aborted traces.

The semantics of an actions system $\mathcal{A}$ is defined as the set of its traces $tr(\mathcal{A})$, which captures the observable behaviors of the system. The set $tr(\mathcal{A})$ can be considered as the disjoint union of infinite, terminated and aborted traces.

2.3.4 Refinement of Unfair Action Systems

A system is defined to implement a specification if every observable behavior of the system is allowed by the specification. Considering that the specification and the implementation of a system can be given as action systems $\mathcal{A}$ and $\mathcal{C}$ respectively, the implementation relation becomes a *refinement* relation between action systems. So, the system $\mathcal{C}$ refines the specification $\mathcal{A}$, noted $\mathcal{A} \sqsubseteq \mathcal{C}$, if and only if the traces of $\mathcal{C}$ can be *observed* in $\mathcal{A}$.

The formal definition of the refinement relation between action systems is given in terms of the *prefix relation* $\preceq$ between traces. Therefore, $\mathcal{A} \sqsubseteq \mathcal{C}$ if and only if, for any trace σ of

[3] The statement $x := y \cdot Q$ denotes a nondeterministic assignment, where x is updated with any value y satisfying Q.

the concrete system $\mathcal{C}$, there exists a trace σ' which *approximates* σ in the abstract system, that is, σ' is either, a prefix of σ and σ' is aborted or $\sigma = \sigma'$ and they are not aborted:

$$\mathcal{A} \sqsubseteq \mathcal{C} \equiv \forall \sigma \in tr(\mathcal{C}) \cdot \exists \sigma' \in tr(\mathcal{A}) \cdot (\sigma' \preceq \sigma \wedge aborted(\sigma')) \vee (\sigma' = \sigma \wedge \neg aborted(\sigma))$$

In practice, the semantical definition of refinement between systems is not used. Refinement verification of actions systems $\mathcal{A}$ and $\mathcal{C}$ is reduced to that of individual actions by a technique known as simulation. Considering a and c as the local variables of $\mathcal{A}$ and $\mathcal{C}$ respectively, x their common variables and $R(a, c, x)$ a data refinement relation, the *forward simulation* between actions A of $\mathcal{A}$ and C of $\mathcal{C}$ is defined as the data refinement $A \leq_R C$. Given that actions take the form of guarded commands $b \rightarrow S$ and $d \rightarrow T$ respectively, where S and T are non miraculous, $b \rightarrow S$ is forward simulated by $d \rightarrow T$ if and only if the following conditions hold:

$$d(x, c)) \wedge R(a, c, x) \leq b(x, a) \tag{2.4}$$

$$R(a, c, x) \wedge d(x, c) \wedge S\,Q(x, a) \leq T\,(\lambda(x, c) \cdot (\exists a' \cdot R(a', c, x) \wedge Q(x, a')))(x, c) \tag{2.5}$$

A very common situation in refinement is when actions of the refined system $\mathcal{C}$ can be divided in two groups: one corresponding to the refinement of abstract actions, and another one to actions refining stuttering steps. Let C and H denote the demonic choice of these group of actions and A the demonic choice of abstract actions. Considering $\mathcal{A}$ and $\mathcal{C}$ as the following action systems

$$\begin{array}{rcl} \mathcal{A} & \widehat{=} & |[\mathsf{var}\, a \bullet P \,;\, \mathsf{do}\, A\, \mathsf{od}]| : x \\ \mathcal{C} & \widehat{=} & |[\mathsf{var}\, c \bullet Q \,;\, \mathsf{do}\, C \,[\!]\, H\, \mathsf{od}]| : x \end{array}$$

the forward simulation of $\mathcal{A}$ by $\mathcal{C}$ under the refinement relation R ($\mathcal{A} \leq_R \mathcal{C}$) is said to hold if the following conditions are verified:

1. $Q(x, c) \leq (\exists a' \cdot R(a', c, x) \wedge P(x, a))$
2. $A \leq_R C$
3. $skip \leq_R H$
4. $R \wedge \neg(gC \vee gH) \leq \neg tA \vee \neg gA$
5. $R \wedge tA \leq t(\mathsf{do}\, H \mathsf{od})$

Condition one states the existence of an abstract state satisfying the initialization of $\mathcal{A}$, which is linked by R to a concrete state where the initialization of $\mathcal{C}$ holds. Condition two and three indicate that the choice of abstract actions A must be data refined by C and that H must refine stuttering actions respectively. Condition four asserts that termination of $\mathcal{C}$ implies terminated or aborted computations of $\mathcal{A}$. Finally, condition five states that termination of A must guarantee that iteration of H terminates.

Example 2.14
In this example the action system PC of example 2.13 is refined. As the example 2.12 of TLA

refinement, a buffer and a new action *cons* are introduced. The refined produced-consumer system becomes

$$PC1 \;\widehat{=}\; |[\mathsf{var}\, C_c, buf \bullet P = D \wedge C_c = \varnothing \wedge buf = \langle\rangle \;;\; \mathsf{do}\; prod \;[]\; cons \;[]\; env\; \mathsf{od}]| : P$$

where

$$\begin{aligned} prod &\;\widehat{=}\; P \neq \varnothing \rightarrow P, buf := P - \{d\}, buf \circ \langle d\rangle \cdot d \in P \\ cons &\;\widehat{=}\; buf \neq \langle\rangle \rightarrow C_c, buf := C_c \cup \{\mathsf{first}(buf)\}, \mathsf{tail}(buf) \\ env_c &\;\widehat{=}\; P = \varnothing \wedge buf = \langle\rangle \rightarrow P, C_c, buf := D, \varnothing, \langle\rangle \end{aligned}$$

In order to prove the correctness of this refinement, that is, that *PC* is forward simulated by *PC1*, the five conditions previously stated must be proved. To do that, the refinement relation is defined:

$$R \mathrel{\widehat{=}} C = C_c \cup \mathsf{ran}(buf)$$

Therefore, $PC \leq_R PC1$ follows from:

- Initialization: $P = D \wedge C_c = \varnothing \wedge buf = \langle\rangle \leq \exists C \cdot (C = C_c \cup \mathsf{ran}(buf) \wedge C = \varnothing \wedge P = D)$.
  This implication follows directly from the premise.

- Main actions: $pc \leq_R prod$ and $env \leq_R env_c$
  Applying (2.4) and (2.5), the proof of $pc \leq_R prod$ follows from [4]

$$R \wedge P \neq \varnothing \leq P \neq \varnothing \tag{2.6}$$

$$\begin{aligned} &R \wedge P \neq \varnothing \wedge \forall d \in P \cdot ([P, C := P - \{d\}, C \cup \{d\}]Q(P, C)) \leq \forall d \in P \cdot \\ &\quad ([P, buf := P - \{d\}, buf \circ \langle d\rangle]\exists C' \cdot (C' = C_c \cup \mathsf{ran}(buf) \wedge Q(P, C'))) \end{aligned} \tag{2.7}$$

  (2.6) follows trivially. (2.7) is equivalent to

$$R \wedge d \in P \wedge Q(P - \{d\}, C \cup \{d\}) \leq \exists C' \cdot C' = C_c \cup \mathsf{ran}(buf \circ \langle d\rangle) \wedge Q(P - \{d\}, C')$$

  This implication follows from the premise considering that $Q(P - \{d\}, C_c \cup \mathsf{ran}(buf) \cup \{d\})$ holds, which is equivalent to the conclusion.

  In a similar way, the proof of $env \leq_R env_c$ follows directly from:

$$R \wedge P = \varnothing \wedge buf = \langle\rangle \leq P = \varnothing \tag{2.8}$$

$$\begin{aligned} &R \wedge P = \varnothing \wedge buf = \langle\rangle \wedge [P, C := D, \varnothing]\, Q(P, C) \leq \\ &\quad [P, C_c, buf := D, \varnothing, \langle\rangle]\, \exists C' \cdot (C' = C_c \cup \mathsf{ran}(buf) \wedge Q(P, C')) \end{aligned} \tag{2.9}$$

- Stuttering actions $skip \leq_R cons$
  Applying the forward refinement definition (2.3), this proof follows from:

$$\begin{aligned} &R \wedge Q(P, C) \wedge buf \neq \langle\rangle \leq \\ &\quad [C_c, buf := C_c \cup \{\mathsf{first}(buf)\}, \mathsf{tail}(buf)]\exists C' \cdot (C' = C_c \cup \mathsf{ran}(buf) \wedge Q(P, C')) \end{aligned}$$

[4] The notation $[x := e]\, Q$, where x and e are lists of the same length of variables and expressions respectively, denotes the predicate Q where all free occurrences of x are replaced by e.

The conclusion of this implication can be rewritten as $\exists C' \cdot (C' = C_c \cup \{\mathsf{first}(buf)\} \cup \mathsf{ran}(\mathsf{tail}(buf)) \wedge Q(P, C'))$ which holds directly from the premise and the definition of R.

- Exit condition: $R \wedge \neg g(prod \,[\!]\, env_c) \wedge \neg g(cons) \leq \neg t(pc \,[\!]\, env) \vee \neg g(pc \,[\!]\, env)$
  This implication holds trivially, remarking that the premise is equivalent to *false*:

$$\neg g(prod \,[\!]\, env_c) \wedge \neg g(cons) \equiv \neg g prod \wedge \neg g env_c \wedge \neg g cons \equiv P = \varnothing \wedge buf \neq \langle\rangle \wedge buf = \langle\rangle$$

- Internal convergence: $R \wedge t(pc \,[\!]\, env) \leq t\,\mathsf{do}\,cons\,\mathsf{od}$
  This proof follows from a well founded argument. In fact, action *cons* decreases the variant $\mathsf{card}(buf)$ which ensures that it cannot be indefinitely executed.

□

2.3.5 Refinement of Fair Action Systems

In [12] the simulation technique of unfair actions systems is extended to consider weak and strong fairness assumptions in the execution of actions. In this section a summary of the weak fairness technique is presented.

Fair Action Systems

A fair action system $\mathcal{A}$ is like an unfair action system where weak or strong fairness assumptions can be associated with each action. Moreover, actions can be nondeterministic. The set of actions associated with weak fairness and strong fairness are denoted by $WF(\mathcal{A})$ and $SF(\mathcal{A})$ respectively; they form a disjoint set of actions. Fair computations $\langle s_0, s_1, \ldots\rangle$ satisfy two conditions:

1. If any action of $WF(\mathcal{A})$ is continuously enabled from some state s_i onwards, then it must be a state s_j, $i \leq j$, where it is executed afterwards.

2. If any action of $SF(\mathcal{A})$ is infinitely often enabled from some state s_i onwards, then it must be a states s_j, $i \leq j$, where it is executed afterwards.

In order to distinguish the actions in $\mathcal{A}$, the labels wf and sf are put before the actions in $WF(\mathcal{A})$ and $SF(\mathcal{A})$ respectively. Actions without labels are executed under minimal progress assumptions: if two or more actions simultaneously are enabled, one of them is nondeterministically chosen and executed. The label of an action A is denoted by l_A.

Fairness is only associated with top level choice; if action A_i is nondeterministic, the fairness assumptions do not apply in the internal choice.

The set of fair computations induce the set of traces $tr(\mathcal{A})$. The semantic definition of refinement is the same that the one used for unfair action systems.

Definition of forward refinement rules for unfair action system requires the notions of termination of weak fair iteration of actions.

Weak Fair Termination

Rules for simulation of fair action systems require notions of termination iteration. Let $\mathcal{A}$ be the fair iteration statement

$$\mathsf{do}\, A_1 \,[\!]\, A_2 \,[\!]\, \ldots \,[\!]\, A_n\, \mathsf{od}$$

To prove the termination of $\mathcal{A}$, a well founded set W and a family of predicates $I = \{I_w \mid w \in W\}$ must be found. Then, it must be proved that all actions of $\mathcal{A}$, executed in a state where I_w holds, do not terminate in a state satisfying I_v where $w < v$ and that some (fair) actions, terminate in a state where I_u holds and $u < w$. The formal definition of termination requires two definitions:

- helpful: $\mathsf{helpful}(I_w, r, A) \mathrel{\widehat{=}} \{I_w \wedge r \wedge tA\}\, A\, \{\exists v \cdot v < w \wedge I_v\}$.
- helpful or stay: $\mathsf{helpful\_or\_stay}(I_w, r, A) \mathrel{\widehat{=}} \{I_w \wedge r \wedge tA\}\, A\, \{\exists v \cdot (v < w \wedge I_v) \vee (I_w \wedge r)\}$.

where r is a predicate characterizing the set of states where A is helpful. Therefore an action A is "helpful" if its execution decreases the index and "helpful or stay" if its execution does not increment it.

Termination under a Precondition An iteration statement $\mathcal{A}$, with weak fair actions set $WF(\mathcal{A}) = \{A_{j_1}, A_{j_2}, \ldots A_{j_k}\}$ terminates under the precondition P if there exists predicates $r_1, r_2, \ldots r_k$ such that the following conditions hold:

1. $P \leq \exists w \cdot I_w$
2. $\mathsf{helpful\_or\_stay}(I_w, r_i, A_j)$ for any $i = 1, \ldots, k$ and any $j \neq J_i$.
3. $\mathsf{helpful}(I_w, r_i, A_{J_i})$ and $I_w \wedge r_i \leq gA_{J_i}$ for any $i = 1, \ldots, k$.
4. $\mathsf{helpful}(I_w, \neg(r_1 \vee r_2 \vee \ldots \vee r_k), A_i)$ for any $i = 1, \ldots, n$.

The soundness of these rules is argued by showing that any state of any computation where I_w holds, for a certain w in W, will eventually lead to a state where I_v holds and $v < w$. From rule one, any initial state where P holds satisfy I_w for a certain w. In any state of any computation where I_w holds, $\exists i \cdot (1 \leq i \leq k \wedge r_i)$ holds or its negation holds. If the negation holds, from rule four, any action of $\mathcal{A}$ is helpful, and therefore decreases the index to establish I_v and $v < w$. If r_i holds, for a certain i in the interval $1..k$, from rule two, execution of any action other than A_{J_i}, does not increment the index and from rule three and the weak fairness assumption, the helpful action A_{J_i}, continually enabled, will be eventually executed and the index w will be decremented.

Commands in $\mathcal{A}$ may contain exit statements which causes the iteration to stop immediately. exit commands can occur at the end of an action or at the end of a branch in an internal choice. To take into account the exit commands in the previous rules, only the definitions of helpful and helpful_or_stay need to be modified:

- $\mathsf{helpful}(I_w, r, A \,;\, \mathsf{exit}) \mathrel{\widehat{=}} \mathsf{helpful\_or\_stay}(I_w, r, A \,;\, \mathsf{exit}) \mathrel{\widehat{=}} true$
- $\mathsf{helpful}(I_w, r, A \,;\, \mathsf{exit} \,[\!]\, A') \mathrel{\widehat{=}} \mathsf{helpful}(I_w, r, A')$
- $\mathsf{helpful\_or\_stay}(I_w, r, A \,;\, \mathsf{exit} \,[\!]\, A') \mathrel{\widehat{=}} \mathsf{helpful\_or\_stay}(I_w, r, A')$

Termination under Continuous Condition An iteration statement $\mathcal{A}$ terminates under a continuous condition P, denoted $[P]\,\mathcal{A}$, if and only if there are no infinite computations from $\mathcal{A}$ such that P always holds. It must be noted that if $\mathcal{A}$ terminates under the precondition P, that is $\{P\}\,\mathcal{A}$ holds, then $[P]\,\mathcal{A}$ follows.

The proof of $[P]\,\mathcal{A}$ can be reduced to an ordinary fair termination. If $\mathcal{A}$ is the iteration statement $\mathsf{do}\, A_1 \,[\!]\, A_2 \,[\!]\, \ldots \,[\!]\, A_n\, \mathsf{od}$ then

$$[P]\,\mathcal{A} \equiv \{true\}\ \mathsf{do}\, P \rightarrow A_1 \,[\!]\, P \rightarrow A_2 \,[\!]\, \ldots \,[\!]\, P \rightarrow A_n\, \mathsf{od}$$

Forward Simulation of Weakly Fair Action Systems

The forward simulation rules consider action systems $\mathcal{A}$ and $\mathcal{C}$ of the form

$$\begin{array}{lcl} \mathcal{A} & \widehat{=} & |[\mathsf{var} a \cdot P \ ;\ \mathsf{do}\, A\, \mathsf{od}]| : z \\ \mathcal{C} & \widehat{=} & |[\mathsf{var} c \cdot Q \ ;\ \mathsf{do}\, C \,[\!]\, H\, \mathsf{od}]| : z \end{array}$$

where

$$\begin{array}{lcl} A & \widehat{=} & A_1 \,[\!]\, A_2 \,[\!]\, \ldots \,[\!]\, A_n \\ C & \widehat{=} & C_1 \,[\!]\, C_2 \,[\!]\, \ldots \,[\!]\, C_m \\ H & \widehat{=} & H_1 \,[\!]\, H_2 \,[\!]\, \ldots \,[\!]\, H_m \end{array}$$

Any A_i, C_i or H_i can be associated with fairness. In order to apply the simulation rules, each concrete action C_i must be decomposed into n internal choices

$$C_i = C_{i,1} \,[\!]\, C_{i,2} \,[\!]\, \ldots \,[\!]\, C_{i,n}$$

such that each abstract action A_j of $\mathcal{A}$ is forward simulated by $C_{i,j}$

$$A_j \leq_R C_{i,j}$$

This decomposition results in the following matrix, where every lower level action simulates the higher level action in the same column:

	A_1	A_2	$\ldots$	A_n
C_1	$C_{1,1}$	$C_{1,2}$	$\cdots$	$C_{1,n}$
C_2	$C_{2,1}$	$C_{2,2}$	$\cdots$	$C_{2,n}$
$\vdots$				
C_m	$C_{m,1}$	$C_{m,2}$	$\cdots$	$C_{m,n}$

It must be noted that the decomposition $C_i = C_{i,1} \,[\!]\, C_{i,2} \,[\!]\, \ldots \,[\!]\, C_{i,n}$ can be verified by refinement calculus. Moreover, some $C_{i,j}$ in the decomposition can be the magic command, because it is the top element in the command lattice and therefore, for any command S, $S = \mathsf{magic} \,[\!]\, S$.

The goal of the forward simulation rules for weak fair action systems is to ensure that, if a higher level action A_i is executed by a fairness assumption, then a subaction $C_{j,i}$ will also be executed in the lower level system. However, as fairness assumptions do not apply to the internal choices, it must be proved that all the subactions will not be infinitely and exclusively executed.

The forward simulation between $\mathcal{A}$ and $\mathcal{C}$, by the data refinement relation $R(a, c, z)$ ($\mathcal{A} \leq_R \mathcal{C}$) follows from the following rules:

1. Initialization: $Q(c,z) \leq \exists a \cdot (R(a,c,z) \wedge P(a,z))$.
2. Main actions: $A_i \leq_R C_{j,i}$ for any i in $1..n$ and j in $1..m$.
3. Stuttering actions: $\mathit{skip} \leq_R H_i$ for any i in $1..k$.
4. Exit condition: $R \wedge \neg(gC \vee gH) \leq \neg tA \vee \neg gA$.
5. Internal convergence: $\{\exists a \cdot R(a,c,z) \wedge tA(c,z)\}\, D$ where

$$D = \mathsf{do}\, l_{C_1} : C_1 \,;\, \mathsf{exit} \,[\!]\, l_{C_2} : C_2 \,;\, \mathsf{exit} \,[\!]\, \cdots \,[\!]\, l_{C_n} : C_n \,;\, \mathsf{exit} \,[\!]\, H_1 \,[\!]\, \cdots \,[\!]\, H_k \,\mathsf{od}$$

6. Fairness condition: $[\exists a \cdot R(a,c,z) \wedge gA_i(a,z)]\, \mathcal{C}^i$ for any $A_i \in WF(\mathcal{A})$, where

$$\begin{array}{rcl} \mathcal{C}^i & = & \mathsf{do}\, C^i_1 \,[\!]\, C^i_2 \,[\!]\, \cdots \,[\!]\, C^i_n \,[\!]\, H_1 \,[\!]\, \cdots \,[\!]\, H_k \,\mathsf{od} \quad \text{and} \\ C^i_j & = & l_{C_j} : (C_{j,1} \,[\!]\, \cdots \,[\!]\, C_{j,i-1} \,[\!]\, C_{j,i} \,;\, \mathsf{exit} \,[\!]\, C_{j,i+1} \,[\!]\, \cdots \,[\!]\, C_{j,n}) \end{array}$$

The first four rules are similar to the corresponding conditions of unfair systems. The internal convergence condition ensures that the number of stuttering steps in the traces of $\mathcal{C}$ are finite unless A is aborting. Finally, the fairness condition guarantees that the low level computations are finite or that a subaction $C_{j,i}$ is executed when a higher level action A_i in $WF(\mathcal{A})$ is continuously enabled.

Strong Fairness The technique to prove the forward simulation of action systems under strong fairness assumptions is similar. However the notion of termination under infinitely often condition is introduced. An iteration statement $\mathcal{A}$ terminates under the infinitely often condition P if and only if there are not infinite computations from $\mathcal{A}$ such that P holds initially and infinitely often after. The proof of this kind of termination is given in terms of Hoare logic by the use of helpful and helpful_or_stay assertions.

2.4 Event B

The event B formal method is an extension of the B method [2], aimed to specify and develop discrete systems. This approach was first published in [7], where the main notion of event system and its methodological guidelines for system development were presented. Subsequent work in [34] proposes a new approach to refinement of event systems closely related to the notion of reachability.

2.4.1 Event B Models

An event B model $\mathcal{S}$, or abstract model, is a state based model which can be defined by the following structure:

$\langle D, x, I(x), U, E\rangle$ where

- D is a list of declarations,
- x is vector of state variables,

- $I(x)$ is a conjunction of predicates known as the invariant,
- U is an initialization statement and
- E is a set of *events*.

The semantics of $\mathcal{S}$ can be explained by a do od construction [24] where events are guarded commands transforming the state variables. Only events with enabled guards are selected for execution, and when two or more events are simultaneously enabled, the choice is done in a non deterministic way.

Events are generalized substitutions borrowed from the B notation. In this section, two forms of events are used:

SELECT $P(x)$ THEN S END

ANY z WHERE $Q(x, z)$ THEN S END

where $P(x)$ and $Q(x, z)$ are predicates, z is a list of fresh variables and S is a non miraculously statement, which can be *skip*, single or simultaneous assignment. The statement S, in an event of type SELECT, can be executed in any state where $P(x)$ holds, while in a statement of type ANY, it can be executed in any state where there exists a value z, such that $Q(x, z)$ holds. These conditions in the execution of S are formalized by the notion of *guard*, which is a predicate associated with any generalized substitution T, indicating the states where execution of T is feasible. Formally, the guard associated with T is given by $grd(T) \equiv \neg[T]\ false$, where $[T]\ R$, for any predicate R, denotes the weakest precondition of T to establish R. So, the guard of events of type SELECT and ANY is as comes next:

$$\begin{array}{lcl} grd(\text{SELECT } P(x) \text{ THEN } S \text{ END}) & \equiv & P(x) \\ grd(\text{ANY } z \text{ WHERE } Q(x, z) \text{ THEN } S \text{ END}) & \equiv & \exists z \cdot (Q(x, z)) \end{array}$$

The semantics of events is given by their weakest preconditions:

$$\begin{array}{lcl} [\text{SELECT } P(x) \text{ THEN } S \text{ END}]\ R & \equiv & P \Rightarrow [S]\ R \\ [\text{ANY } z \text{ WHERE } Q(x, z) \text{ THEN } S \text{ END}]\ R & \equiv & \forall z \cdot (Q \Rightarrow [S]\ R) \end{array}$$

where z is not free in the postcondition R of the second line, and the weakest precondition $[S]\ R$ depends on the statement S:

- If S is *skip*, $[S]\ R$ is equivalent to R.
- If S is a single assignment $x := e$, $[S]\ R$ is equivalent to R where any occurrence of x is replaced by e.
- If S is a multiple assignment $x_1, \ldots x_n := e_1 \ldots e_n$, $[S]\ R$ is equivalent to R where all occurrences of the variables in the list $x_1, \ldots x_n$ are simultaneously replaced by the corresponding expressions in the list $e_1, \ldots e_n$.

An abstract model $\mathcal{S}$ is well formed if it fulfills two conditions:

- The initialization statement establishes the invariant: $D \Rightarrow [U]\ I$.
- Execution of any event e in E always terminates and it preserves the invariant: $D \wedge I \Rightarrow [e]\ I$.

For ever running systems, an extra condition is deadlock-freeness: $D \wedge I \Rightarrow \exists e \cdot (e \in E \wedge grd(e))$. However, this condition obviously does not hold in the last state of a terminating system.

Example 2.15
In this example the abstract producer-consumer system, treated in examples 2.5, 2.7 and 2.13 with other formalisms, is specified in the event B notation. The goal of this system is to transfer any data from the producer set P to the consumer set C; this behavior is achieved by the event *pc*. Event *env* resets the variables to their initial values to model another production-consumption cycle. In this example, the invariant only specifies the type of the variables.

SYSTEM PC SETS D VARIABLES P, C	INVARIANT $C \subseteq D \wedge P \subseteq D$ INITIALISATION $P, C := D, \varnothing$ EVENTS	$pc \mathrel{\widehat{=}}$ ANY d WHERE $d \in P$ THEN $C, P := C \cup \{d\}, P - \{d\}$ END ;	$env \mathrel{\widehat{=}}$ SELECT $P = \varnothing$ THEN $P, C := D, \varnothing$ END

□

2.4.2 Modalities

Invariant preservation allows the specification of safety properties in abstract models. However, a new notation is required to specify liveness properties in event B along with the corresponding proof obligations for their verification.

In [7] news constructs are introduced to specify liveness properties or modalities. The main construct to specify liveness properties is presented in two forms:

SELECT P LEADSTO Q WHILE E_1 OR $\cdots$ OR E_n INVARIANT J VARIANT V END	ANY y WHERE P LEADSTO Q WHILE E_1 OR $\cdots$ OR E_n INVARIANT J VARIANT V END

where P, Q and J are predicates, V is a variant function and $E_1, \ldots, E_n$ are events of the system. The construct with the keyword ANY allows a nondeterministic choice by the variable y, it generalizes the construct with the keyword SELECT . Informally, this kind of modality specifies that iteration of the statement $E_1 \,[\!]\, \ldots \,[\!]\, E_n$ in a state where P holds, terminates into a state where Q holds. Moreover, it also states that J is invariant in the iteration of events and that execution of any event E_i decrements the variant V. In other words, in a TLA style, this construct specifies the liveness property $P \rightsquigarrow Q$ and it gives information to verify it.

The proof obligations required to verify the construct with the keyword ANY are as follows:

$$\forall y \cdot (P \Rightarrow J)) \tag{2.10}$$
$$\forall y \cdot (P \Rightarrow \forall z \cdot (I \wedge J \Rightarrow V \in \mathbb{N})) \tag{2.11}$$
$$\forall y \cdot (P \Rightarrow \forall z \cdot (I \wedge J \wedge \neg Q \Rightarrow [E_i]\, J)) \tag{2.12}$$
$$\forall y \cdot (P \Rightarrow \forall z \cdot (I \wedge J \wedge \neg Q \Rightarrow [v := V]\, [E_i]\, (V < v))) \tag{2.13}$$
$$\forall y \cdot (P \Rightarrow \forall z \cdot (I \wedge J \wedge \neg Q \Rightarrow grd(E_1) \vee \cdots \vee grd(E_n))) \tag{2.14}$$

In these proof obligations, the quantified variable z denotes the state variables modified in the iteration. The first four conditions are in fact the proof that the iteration of $E_1 \,[\!]\, \ldots \,[\!]\, E_n$ terminates when its execution starts in a state satisfying P. The last proof obligation ensures that upon termination, the iteration of events establishes Q. The proof obligations to verify the construct with the keyword SELECT are similar, only the universal quantification over y disappears.

It must be noted that the set of natural numbers $\mathbb{N}$ is only one possibility. In fact, any well founded set can be used in these proof obligations with its corresponding order relation.

Example 2.16
This example presents the specification and proof of a dynamic property holding in the producer-consumer system of example 2.15. The property states that any data d in the producer set P will be eventually transfered to the consumer set C.

ANY d WHERE
$d \in P$
LEADSTO
$d \in C$
WHILE
pc OR env
INVARIANT
$true$
VARIANT
$\mathsf{card}(P)$
END

The invariant J of this modality is the predicate $true$ and the variant V is the number of elements in P. Taking invariant I and events pc and env of system PC in example 2.15, the proof obligations verifying this property are:

$$d \in P \Rightarrow true$$
$$P \subseteq D \wedge C \subseteq D \wedge d \in P \Rightarrow \mathsf{card}(P) \in \mathbb{N}$$
$$P \subseteq D \wedge C \subseteq D \wedge d \in P \Rightarrow [pc \,[\!]\, env]\ true$$
$$P \subseteq D \wedge C \subseteq D \wedge d \in P \wedge d \notin C \wedge \mathsf{card}(P) = v \Rightarrow [pc \,[\!]\, env]\ \mathsf{card}(P) < v$$
$$P \subseteq D \wedge C \subseteq D \wedge d \in P \wedge d \notin C \wedge \mathsf{card}(P) = v \Rightarrow grd(pc) \vee grd(env)$$

The first two proof obligations hold trivially. Given that pc and env are always terminating events, preservation of the invariant follows. Considering that event pc removes elements from P, the decrement of the variant follows for this event. On the other hand, under the premise $d \in P$, the guard of event env does not hold, and therefore it becomes miraculous under this condition and the fourth proof obligation holds. Finally, the last proof obligation follows from the fact that $d \in P$ imply the guard of event pc.

□

2.4.3 Refining Event B Models

In event B, abstract models are gradually implemented by a sequence of refinement steps. In each refinement step, a new concrete model is developed, which simulates the abstract one. The concrete model reduces nondeterminism of the abstract model by strengthening the guard of the concrete events. Moreover, it allows the implementation of abstract structures

into concrete variables, and makes possible the introduction of new events. The concrete, or refined model, is a structure similar to the abstract one, with list of declaration D', state variable(s) y, invariant J and set of events E' which is the union of two disjoint sets: the set of refined events $\mathcal{R}$ and the set of new events $\mathcal{H}$.

In order to guarantee the correctness of a refinement, the invariant J, or *gluing invariant*, must relate the variable x of the abstract model with the concrete one y. Moreover, the initialization statement and events in E' must satisfy the following conditions:

- Initialization: $D \wedge D' \Rightarrow [U'] \neg [U] \neg J$.
- Refined Events: $\forall e \cdot (e \in E \wedge I \wedge J \Rightarrow [e'] \neg [e] \neg J)$ where e' denotes the refinement of e in $\mathcal{R}$.
- New events: $\forall h \cdot (h \in \mathcal{H} \wedge I \wedge J \Rightarrow [h] J)$.
- Bounded *skip*: $\forall h \cdot (h \in \mathcal{H} \wedge I \wedge J \wedge V = n \Rightarrow [h] V < n)$.
- Deadlock-freeness: $I \wedge J \wedge \exists e \cdot (e \in E \wedge grd(e)) \Rightarrow \exists e' \cdot (e' \in E' \wedge grd(e'))$

The initialization condition states that U, the initialization statement, establishes the existence of an abstract value x satisfying $I(x)$, which is related to the concrete variable y by the gluing invariant $J(y, x)$. The refined events condition ensures that any postcondition established by any abstract event e, is established by the corresponding refined event e' as well. The new events condition guarantees that any new event refines *skip*; it allows the refinement of stuttering steps in the abstraction. The bounded *skip* condition ensures that iteration of new events is finite and therefore the refined system executes eventually a refined event. Finally, the deadlock-freeness condition states that the concrete model does not introduce more deadlocks than the abstract one.

Refining Modalities

In order to guarantee the preservation of modalities in the refined system, all proof obligations concerning the verification of modalities must be preserved. Proof obligations (2.10) and (2.11) are not concerned by refinement. Proof obligations (2.12) and (2.13) are preserved by the very definition of refinement. However, there is no guarantee that (2.14) holds in the refinement, because the concrete guards are stronger than the abstract ones.

The main goal of proof obligations (2.10–2.14) is to guarantee that iteration of abstract events terminates. Nevertheless, the concrete system contains not only refined events, but new events as well. Refined events cannot take the control for ever because (2.13) is preserved and therefore their iteration terminates. New events, by the bounded *skip* condition in the refinement, do not run for ever either. In this way, only must be ensured that the iteration of refined and new events does not terminate before its normal abstract end. However, this is guaranteed by the deadlock-freeness condition of the refinement. In fact, the bounded *skip* and deadlock-freeness conditions in the refinement are introduced to guarantee the preservation of liveness properties from the abstract model.

Example 2.17

A refinement of the abstract producer-consumer system PC from example 2.16 is [5]:

SYSTEM

$PC1$

REFINES

PC

VARIABLES

P, C_c, buf

INVARIANT

$C_c \subseteq D \wedge buf \subseteq D \wedge$
$C = C_c \cup buf$

INITIALISATION

$P, C_c, buf := D, \varnothing, \varnothing$

EVENTS

$prod \mathrel{\widehat{=}}$
SELECT
$d \in P$
THEN
$P, buf := P - \{d\}, buf \cup \{d\}$
END ;
$cons \mathrel{\widehat{=}}$
ANY d WHERE
$d \in buf$
THEN
$C_c, buf := C_c \cup \{d\}, buf - \{d\}$
END ;
$env_c \mathrel{\widehat{=}}$
SELECT
$P = \varnothing \wedge buf = \varnothing$
THEN
$P, C_c, buf := D, \varnothing, \varnothing$
END

As in examples 2.5, 2.12 and 2.14, a buffer is introduced between the producer and the consumer; however in this example, the buffer is simply modeled by a set and it is not a sequence as in the previous examples. The abstract consumer set C is implemented as the union of the concrete set C_c and the buffer buf; this is specified by the gluing invariant $C = C_c \cup buf$.

In order to prove the correctness of this refinement, the five proof obligations stated in the section 2.4.3 must be proved. Given that the initialization condition is easily discharged, only the last four proof obligations are commented [6].

Refined events The proof of the refinement of env by env_c is trivial. The refinement of pc by $prod$ follows from

$$C = C_c \cup buf \wedge d \in P \Rightarrow \exists d' \cdot (d' \in P \wedge C \cup \{d\} = C_c \cup buf \cup \{d'\})$$

New events The refinement of $skip$ by $cons$ follows from

$$C = C_c \cup buf \wedge d \in buf \Rightarrow C = (C_c \cup \{d\}) \cup (buf - \{d\})$$

Bounded $skip$ $cons$ decrements the number of elements in buf. It follows from

$$\mathsf{card}(buf) = v \wedge d \in buf \Rightarrow card(buf - \{d\}) < v$$

[5] For the sake of simplicity, the events pc and env in the abstraction are renamed to $prod$ and env_c respectively in this refinement.

[6] All of this proof obligations are under the assumption of the abstract invariant $P \subseteq D \wedge C \subseteq D$.

Deadlock-freeness This condition follows from the calculations of guards:

$$grd(pc) \equiv grd(prod) \equiv \exists d \cdot (d \in P)$$
$$(grd(env) \equiv P = \varnothing) \wedge grd(env_c) \equiv (P = \varnothing \wedge buf = \varnothing)$$

Finally, the bounded *skip* and the deadlock-freeness condition guarantee preservation of the modality specified in the example 2.16. □

2.4.4 Refinement and Reachability

In [34] a new approach to verification of modalities is proposed which is closely related to refinement. The approach does not use particular constructs to state explicitly modalities and they are implicitly specified by the set of system behaviors.

In the approach, behaviors of abstract models are partially specified. Events of the abstract model only generate selected behaviors describing a main task, performed in *one shot*. Behaviors which are not of interest are ignored at this stage. Refinements of the abstract model gradually introduce new events, called *anticipating events*, generating complementary behaviors. Anticipating events modify abstract variables and therefore do not refine *skip*. However, in order to ensure that anticipating events do not take the control forever, they must decrement a variant. Moreover, anticipating events must not modify variants associated with anticipating events introduced in early refinement stages. Events generating complementary behaviors can be merged to conform to the final model. The technique is illustrated in the following example.

Example 2.18
The goal of this example is to show that refinement *PC1* in example 2.17 respects the dynamic property:

Any data d in the buffer buf is eventually transfered to the consumer set C_c.

The argument to verify this property is as comes next.

The first step is to consider any data d in D. The abstract model only considers the following five events:

prod1 $\widehat{=}$
SELECT
$d \in P$
THEN
$P := P - \{d\} \parallel$
$buf := buf \cup \{d\}$
END ;

prod2 $\widehat{=}$
ANY d' WHERE
$d' \in P \wedge d \notin buf \wedge d' \neq d$
THEN
$P := P - \{d'\} \parallel$
$buf := buf \cup \{d'\}$
END ;

cons1 $\widehat{=}$
SELECT
$d \in buf$
THEN
$C_c := C_c \cup \{d\} \parallel$
$buf := buf - \{d\}$
END ;

cons2 $\widehat{=}$
ANY d' WHERE
$d' \in buf \wedge d \notin buf$
THEN
$C_c := C_c \cup \{d'\}$
$buf := buf - \{d'\}$
END ;

env $\widehat{=}$
SELECT
$P = \varnothing \wedge buf = \varnothing$
THEN
$P, C_c, buf := D, \varnothing, \varnothing$
END

Event env_c, as before, is used to reset the variables to their initial values. Event *prod1* specifies a behavior where the data d is transfered from P to *buf* and event *prod2* transfers, in the same way, any other data d' different from d only when d is not in the buffer. On another hand, when the data d is in the buffer, event *cons1* ensures that d will be transfered from *buf* to C_c and event *cons2* ensures a similar transfer for any other d' only when d is not in the buffer. In this way, when d arrives to the buffer, *cons1* is the only enabled event and it guarantees the transfer of d from *buf* to C_c in one shot. When d is not in the buffer, events *prod2* and *cons2* transfer from P to C_c any other data d' distinct from d. In the following refinements, two more events are added to the model to consider complementary behaviors.

In the following refinement, *prod3* is introduced as a new anticipating event. It transfers data from P to *buf* when d is already in the buffer. This event cannot take the control forever because its execution is bounded by the number of elements in P. That is, the variant associated with *prod3* is $\mathsf{card}(P)$.

prod3 $\widehat{=}$
ANY d' WHERE
$d' \in P \wedge d \in buf$
THEN
$P, buf := P - \{d'\}, buf \cup \{d'\}$
END ;

The last refinement introduces the new anticipating event *cons3*. The goal of this event is to allow behaviors where any data d', distinct from d, is transfered from *buf* to C_c when the data d is in the buffer. Execution of *cons3* is bounded by the variant $\mathsf{card}(buf)$ guarantee that this event does not take the control forever.

cons3 $\widehat{=}$
ANY d' WHERE
$d' \in buf \wedge d \in buf \wedge d' \neq d$
THEN
$C_c, buf := C_c \cup \{d'\}, buf - \{d'\}$
END

The last step is to merge events to get the final model. The merge of events *prod1*, *prod2* and *prod3* gives as result the event *prod* of system *PC1*, and from the merge of events *cons1*, *cons2* and *cons3*, the event *cons* is obtained.

□

2.5 Fairness

In this section we review some related works having studied fairness in the context of program verification, program development, semantics or implementation of parallelism.

In [8, 9] an approach to program transformation is proposed, where weak or strong fair parallelism is reduced to ordinary parallelism. Apart from giving insight on fairness implementation, the transformation allows applying an extension of the proof system of Owicki-Gries [51] to the verification of programs. The transformation embeds a scheduler in the program, modeled by random assignment, which guarantees fair computations. Implementation of random assignment can provide concrete scheduling policies as round robin or queuing.

A classical study on fairness is presented in [28]. The main concern of this work is the fair termination of iteration of guarded commands under unconditional, weak or strong fairness assumptions. Ideas coming from different sources are uniformly presented. Several rules are presented to prove total correctness of loops by using Hoare's triplets. Soundness and relative completeness of that rules are proved. Apart from traditional fairness assumptions, the notions of *equifairness* and *extreme* fairness are introduced. Equifair schedulers try to give an equally fair chance to each guard in a group of jointly enabled guards. Extreme fairness

deals with loops where probabilities are attached to its guarded commands. A predicate transformer for strong fair termination of loops is presented and it is related to the rules for proving the total correctness of loops under strong fairness.

Fairness has been clearly defined in temporal logic, as well as different sorts of safety and liveness properties [44]. In [43] a proof system, founded on an assertional reasoning, is proposed to verify reactive or concurrent programs. In the context of verification of temporal properties, model checking is a useful technique that can be applied in the case of finite-state systems. In [27] fairness assumptions are used to reduce the combinatorial explosion in the verification of abstract transition systems by model checking.

Computational models, for concurrent program development, can be represented by the do od construction of guarded commands. Different works can be distinguished according to the guards of the commands. The following paragraphs give a brief survey of formalisms where fairness plays a fundamental role.

The computational model of UNITY [21] can be modeled by a do od construction where the guard of the commands is always true. These statements are executed according to an "unconditional fairness" assumption, which indicates that in any computation any statement is infinitely often executed. Liveness properties are specified and proved by two relations on state predicates known as *ensures* and *leads-to*. *ensures* relation is defined by the Hoare triplets quantified over the statements of a program; the fairness assumption guarantees the execution of the helpful statement performing the basic progress specified by this relation. *leads-to* is defined as the transitive and disjunctive closure of *ensures*.

Following the computational model of UNITY, in [22] a calculus of predicate transformers is proposed to reason about safety and liveness properties. A program is a set of variables and a set of predicate transformers denoting the weakest precondition of non miraculous, always terminating and bounded nondeterministic statements. Progress properties are defined by a predicate transformer defined in [39] which is the strongest solution of a set of equations where the *ensures* relation forms the base case. In [46], the study of composition of *leads-to* properties is pursued. In this approach, the predicate transformers defining *ensures* and *leads-to* properties proposed in [36] are used to develop a general theory of composition of *leads-to*. These predicate transformers are defined by the fixpoints of equations relating the weakest precondition of statements in a program.

More general computation models, like those of TLA or Action Systems, where the guards of commands in the do od construction are state predicates, allow different definitions of fairness like weak or strong fairness. An early proposal of a predicate transformer, defined by fixpoint equations, characterizing the weakest precondition of do od constructions under weak fairness assumptions is presented in [52].

In [37], a methodology for designing proof systems for *leads-to* properties under various fairness assumptions is proposed. This work shows how the specification and proof of liveness properties in UNITY can be adapted to reason about different kinds of fairness. The methodology characterizes different kinds of fairness by CTL* formulae. These formulae are then characterized by fixpoint in the μ-calculus. Finally these fixpoint characterizations are used to propose various definitions of the *ensures* relation according to fairness assumptions.

2.6 Conclusion

We have reviewed the main approaches to specification and refinement of systems in state based formalisms. The approaches emphasizes two complementary views of a system:

- one considering the system as a set of properties in a particular logic and
- another one regarding the system as an algorithm in a programming-like notation.

TLA is a classical example where a system is considered as a set of safety and liveness properties and Action System illustrates the view of a system in a programming like notation. UNITY propose integration of these views in the system development. However, the algorithmic aspect, that is the UNITY program, appears in the last step of the development in a sudden way and it is not the result of a calculus as in Action System. In event B, systems are developed by a refinement calculus and, moreover, certain safety and liveness properties can be specified and proved.

On another hand, from the previous section, we can observe two main approaches where fairness is considered in the formal reasoning of systems:

- fair termination of iteration of actions and
- temporal reasoning characterizing fair computation sequences.

Approaches like [8, 9, 28, 10], where the notion of fair termination of iteration is used to reason about fairness, provide proof rules which can be directly applied in the code of models. These rules take advantage of the decrement of variants to ensures termination and they are difficult to apply in practice, due to the lack of modularity. This situation contrasts with the approaches using temporal logic, like TLA or temporal logic as in [43, 45], which provides more flexible rules to reason about liveness under fairness constraints. However, the application of these rules in the verification of algorithms, at the implementation level, as it is often the case, becomes very difficult; as an example of theses difficulties we can cite the verification of the Ricart-Agrawala mutual exclusion algorithm [66].

In this thesis we are interested in state based formalisms from a practical point of view. Therefore we are interested in the algorithmic approach of systems as it is done in Action Systems or event B. From our practical point of view, the main advantage of event B over Action Systems is the existence of automatic tools for the development of models; for this reason we use event B as the framework of our study. However, as we have already explained, the specification and proof of liveness properties in event B is not as flexible as those of the logical approaches. For this reason we propose the incorporation of temporal logic in the specification and proof of liveness properties in event B. We are aware that the logical system of TLA is more flexible than that of UNITY, but the logic programming of UNITY is more suited to our purpose for following reasons:

- Liveness properties are inductively defined from a basic relation.
- The basic relation is defined by quantification of triplets.
- The definition of the basic relation isolates fairness assumptions.

The inductive definition of liveness properties allows modular proofs by application of a reduced number of rules. Defining the basic relation by triplets admits simple redefinitions to adapt it to the particularities of event B and allows the verification of basic properties by the "Click'n Prove" prover [4]. Moreover, the isolation of fairness in the definition of the basic relation gives the possibility of accommodate different fairness assumptions in the reasoning of liveness properties.

We take advantage of temporal logic to specify and prove liveness properties under fairness assumptions, but we do not interpret temporal formulas by state sequences as it is traditionally done. Instead, we use the notion of termination of iteration advocated above. In this way, we formally relate the notion of *reachability*, as specified by temporal formulas with the notion of *termination* of iteration under minimal progress and weak fairness assumptions.

In the following chapters of this thesis we develop our approach: we propose proof rules to verify basic properties in a UNITY-like style under minimal progress and weak fairness assumptions in event B models. That allows the specification and proof of general liveness properties under these constraints. Then, we propose rules to demonstrate the preservation of liveness properties under refinement with minimal progress and weak fairness assumptions. The semantics of liveness properties is given in a set theoretical framework. It provides the proof of soundness of our rules. The approach is illustrated by a series of examples.

Chapter 3

Liveness Properties in Event B

Résumé

Dans ce chapitre nous présentons les aspect pratiques de notre recherche. Nous montrons comment spécifier et prouver des propriétés de vivacité dans le style UNITY *sous les hypothèses de progrès minimal et d'équité faible. Nous donnons des obligations de preuve pour prouver des propriétés de base sous ces hypothèses d'équité. Par différents exemples nous montrons comment prouver des propriétés de vivacité générales. Nous présentons des règles qui sont des conditions suffisantes pour préserver dans les raffinements les propriétés de vivacité sous les hypothèses de progrès minimal et d'équité faible. L'application de ces règles est aussi illustrée par des exemples.*

In this chapter we present the specification, proof and refinement of liveness properties in event B systems. The properties are specified in a notation borrowed from the UNITY logic. The proofs of these properties and their preservation are given by UNITY theorems and by proof obligations, in the predicate transformer calculus, specially designed for event B systems, under weak fairness and minimal progress assumptions.

This chapter is structured in four sections. The first section gives general notions of liveness properties in an UNITY-like style, where liveness properties are separated in basic and general properties. Section two and three present proof obligations for verification of basic properties under a minimal progress and weak fairness assumptions respectively. Section four proposes rules for the preservation under refinement of basic properties considering the minimal progress and weak fairness assumptions. These sections present some examples to illustrate the approach; these examples are entirely proved with the "Click'n Prove" prover [4].

3.1 Liveness Properties in a UNITY-like Style

Liveness properties are separated in two classes: basic and general properties. These properties are specified by *ensures* and *leads-to* state predicates relations respectively.

3.1.1 Basic Liveness Properties

Basic properties specify transitions made by the execution of an atomic event. A basic property is specified by an *ensures* relation, denoted by $\gg$ in this thesis.

Informally, for any state predicates P and Q, a system satisfies the property $P \gg Q$, if the execution of one event eventually makes a transition from a state where P holds into another one where Q holds and while Q is *false*, P is *true*. Given the nondeterministic computation model of systems, the execution of a particular event cannot be guaranteed if fairness assumptions are not adopted. Therefore, the definition of the *ensures* relation takes into account fairness assumptions, and the symbols $\gg_w$ and $\gg_m$ are used in this thesis to stand for the *ensures* relation under weak fairness and minimal progress assumptions respectively.

On another hand, inspired from [62], definitions of *ensures* relations take into account the notion of *strongest invariant*. It is defined as the strongest predicate which is both preserved by all events and implied by the initial conditions of the system. Incorporating the strongest invariant in definitions of basic properties, allows the elimination of the substitution axiom from the programming logic, while the substitutions of invariants by *true* continue to apply [53]. Formally, the strongest invariant is defined as comes next:

Definition 3.1 (Strongest Invariant)
For event system $\mathcal{S}$ with state variable x, initialization statement U and set of events E, the strongest invariant SI is defined as the strongest predicate X satisfying

$$(X \Rightarrow [S]\, X) \wedge (([x' := x]\ \mathsf{prd}(U)) \Rightarrow X)$$

where S is the bounded choice of events $[]_{e \in E}\ e$ and $\mathsf{prd}(U)$ is the after-before relation associated with U.

In sections 3.3 and 3.2, the strongest invariant SI is used to formally define the *ensures* relations in event B systems under weak fairness and minimal progress assumptions

3.1.2 General Liveness Properties

General liveness properties are specified by the *leads-to* relation denoted by the symbol $\leadsto$. Informally a property $P \leadsto Q$ specifies that the system eventually reaches a state satisfying Q whenever it reaches any state in P. There are three basic differences between a $\leadsto$ relation and a $\gg$ relation. The first difference is the number of steps involved in the transition from P to Q. With $\gg$, the transition is done by the execution of an atomic event. However, with $\leadsto$, the number of atomic transitions is not specified. The second difference is that $P \gg Q$ asserts that the system maintains P as long as Q is not established, while this assertion is not guaranteed by the property $P \leadsto Q$. Finally, the third difference is that a general liveness property does not depend directly on any fairness assumption as a basic liveness property does. Fairness assumptions are isolated in the definition of the basic relation, and other fairness assumptions can be taken into account in the definition of the *ensures* relation, as it is done in [37].

A property $P \leadsto Q$ holds in a event B system, if it is derived by a finite number of applications of the rules defined by the UNITY theory:

	ANTECEDENT	CONSEQUENT
BRL	$P \gg Q$	$P \rightsquigarrow Q$
TRA	$P \rightsquigarrow R$, $R \rightsquigarrow Q$	$P \rightsquigarrow Q$
DSJ	$\forall m \cdot (m \in M \Rightarrow P(m) \rightsquigarrow Q)$	$\exists m \cdot (m \in M \wedge P(m)) \rightsquigarrow Q$

where M in the disjunction rule DSJ is any index set. It is remarked that the *ensures* relation $\gg$ in the basic rule BRL, must be instantiated to $\gg_w$ or $\gg_m$ to derive general liveness properties under weak fairness or minimal progress assumptions.

So as to reason about liveness properties, the proof system of UNITY is incorporated in the framework of B event systems. Moreover, all theorems in [21] concerning *ensures* and *leads to* relations, can be used in the proof of several properties of B event systems. In this thesis, the following theorems of UNITY are widely used:

	ANTECEDENT	CONSEQUENT
IND	$\forall v \cdot (P \wedge V = v \rightsquigarrow P \wedge V < v \vee Q)$	$P \rightsquigarrow Q$
CAN	$P \rightsquigarrow O \vee Q, O \rightsquigarrow R$	$P \rightsquigarrow R \vee Q$
SCJ	$P \rightsquigarrow Q$, $\quad R$ is a boolean constant expression	$P \wedge R \rightsquigarrow Q \wedge R$

In IND, the "induction rule", V is a variant function over a well founded set, and v is universally quantified over that set. IND rule indicates that in any state where P holds and the variant value is v, if the execution of a program changes into a state where P holds but the value of the variant is lower than v or Q holds, then the property $P \rightsquigarrow Q$ holds in the system. The CAN rule is named the "cancellation theorem". In SCJ, the "stable conjunction" rule, the boolean constant expression R is any constant expression containing constants and free variables.

It must be noted that specifications and proofs of liveness properties with the *leads-to* relation are given in the framework of the proof system of UNITY, while the proof of properties with the *ensures* relation are given in a B framework as it is indicated in the following sections.

3.2 Specification under Minimal Progress

A minimal progress assumption in a nondeterministic B event system does not guarantee the execution of a particular event. In fact, if two or more events are simultaneously enabled, one of them is selected for execution in a nondeterministic way. When an abstract model needs to specify execution of a particular event, the system must code *abstract schedulers* [9] in order to guarantee its execution.

A basic liveness property under a minimal progress assumption takes the general form $P \gg_m Q$. In order to satisfy this kind of property, a B event system must guarantee that Q eventually holds by the execution of one event when the system reaches a state where $P \wedge \neg Q$ holds. However, under a minimal progress assumption, the only one possibility to satisfy this property is to impose that any event must be able to establish Q and that, at least, one of them must be enabled when the system reaches a state where $P \wedge \neg Q$ holds. This reasoning allows the definition of basic properties under minimal progress assumptions:

Definition 3.2 (ensures under minimal progress)
For any B *event system* $\mathcal{S}$ *with set of events* E *and state predicates* P *and* Q,

$$P \gg_m Q \mathrel{\widehat{=}} SI \wedge P \wedge \neg Q \Rightarrow (([S]\ Q) \wedge grd(S))$$

where S is the bounded choice $[]_{e\in E}\ e$.

It must be remarked in definition 3.2 that $[S]\ Q$ is equivalent to the conjunction of weakest preconditions $\forall e \cdot (e \in E \Rightarrow [e]\ Q)$ and $grd(S)$ is equivalent to the disjunction of guards $\exists e \cdot (e \in E \wedge grd(e))$.

In practice, definition 3.2 is difficult to apply in the verification of basic liveness properties, because the strongest invariant needs to be calculated. However, in order to dispose of proof rules which can be used in practice, the invariant I of the event system $\mathcal{S}$ can be used instead of the strongest invariant SI. Therefore the following rules are sufficient to prove basic liveness properties under minimal progress assumptions, taking into account the declarations of definition 3.2:

	ANTECEDENT	CONSEQUENT
MP0	$I \wedge P \wedge \neg Q \Rightarrow [S]\, Q$	$P \gg_m Q$
MP1	$I \wedge P \wedge \neg Q \Rightarrow grd(S)$	

In order to demonstrate that MP0 and MP1 are sufficient conditions to guarantee basic liveness properties, it suffices to note that $SI \Rightarrow I$.

In spite of the fact that MP0 rule states that any event in a system must be able to establish the predicate Q, it must be noticed that under the assumption $P \wedge \neg Q$ of that rule, some events can be miraculous. That is, miraculous events are able to establish any postcondition. In the following example a miraculous event is presented in the proof of a basic liveness property.

Example 3.1
In a UNITY-like style, the producer-consumer system PC of example 2.15, in page 31, satisfies for any data item d in D, the general liveness property

$$d \in P \rightsquigarrow d \in C$$

stating that any data d in the producer set P will eventually be transferred to the consumer set C. In order to prove this property under a minimal progress assumption, it must be remarked that any data d will be eventually in the consumer set C because system PC cannot indefinitely stay in a state where both d remains in P and the cardinality of P does not decrease. This fact can be specified by:

$$d \in P \wedge \mathsf{card}(P) = n \rightsquigarrow (d \in P \wedge \mathsf{card}(P) < n) \vee d \in C \tag{3.1}$$

Then $d \in P \rightsquigarrow d \in C$ follows by application of the induction rule IND to (3.1) where the variant V of the rule is $\mathsf{card}(P)$. Finally, this last modality is derived from the following basic liveness property, by application of the BRL rule:

$$d \in P \wedge \mathsf{card}(P) = n \gg_m (d \in P \wedge \mathsf{card}(P) < n) \vee d \in C$$

Under a minimal progress assumption, this property states that execution of any event of PC when d is in P, terminates into a state where d continues to stay in P but the cardinality of P decreases or d is transferred to C. The basic property is proved by rules MP0 and MP1. In order to verify MP0, two proofs are required:

$$d \in P \wedge \mathsf{card}(P) = n \Rightarrow [pc]\,(d \in P \wedge \mathsf{card}(P) < n) \vee d \in C$$
$$d \in P \wedge \mathsf{card}(P) = n \Rightarrow [env]\,(d \in P \wedge \mathsf{card}(P) < n) \vee d \in C$$

The first implication follows directly by calculating the weakest precondition of event *pc*:

$$\forall e \cdot (e \in P \wedge d \in P \wedge \mathsf{card}(P) = n \Rightarrow (d \in P - \{e\} \wedge \mathsf{card}(P - \{e\}) < n) \vee d \in C \cup \{e\})$$

The second implication holds trivially considering that event *env* is miraculous under the assumption $d \in P$; in fact, the guard of event *env* is $P = \varnothing$ and its conjunction with $d \in P$ is *false*. On another hand, the proof of WF1 follows from:

$$d \in P \wedge \mathsf{card}(P) = n \Rightarrow (grd(pc) \vee grd(env))$$

The proof follows by considering that $d \in P$ implies $\exists d \cdot (d \in P)$, which is the guard of *pc*.

□

The proof of general liveness properties does not necessarily requires the use of variant functions. In the following example a simple system is specified where the liveness proof does not require a variant function.

Example 3.2
In this example, a simple mutual exclusion system *MutexSeq* is presented. The system is made up of set of processes *PR* which access a critical section. Each process can be in one of three states: *idle* (*ID*), *waiting* (*WT*) or *active* (*AC*). Processes in state *idle* do not ask for access to the critical section. When a process request access, its state changes to *waiting*. A waiting process changes to state *active* when it accesses the critical section. Finally, an active process returns to the state *idle* when it leaves the critical section. System *MutexSeq* is specified as follows:

SYSTEM
MutexSeq
SETS
PR;
$ST = \{ID, WT, AC\}$
CONSTANTS
pid
PROPERTIES
$pid \in PR \rightarrowtail\!\!\!\!\!\rightarrow 0..\mathsf{card}(PR) - 1 \wedge$
$\mathsf{card}(PR) > 1$
DEFINITIONS
$Idl \mathrel{\widehat{=}} st^{-1}[\{ID\}]$;
$Wtg \mathrel{\widehat{=}} st^{-1}[\{WT\}]$;
$Act \mathrel{\widehat{=}} st^{-1}[\{AC\}]$;
VARIABLES
st, pt
INVARIANT
$st \in PR \rightarrow ST \wedge$
$pt \in 0..\mathsf{card}(PR) - 1 \wedge$
$\mathsf{card}(Act) \leq 1 \wedge$
$\forall p \cdot (p \in PR \wedge p \notin Act \wedge pid(p) = pt \Rightarrow Act = \varnothing) \wedge$
$\forall p \cdot (p \in Act \Rightarrow pid(p) = pt)$

INITIALISATION
$st, pt := PR \times \{ID\}, 0$
EVENTS
req $\mathrel{\widehat{=}}$
ANY p WHERE
$p \in Idl \wedge pid(p) = pt$
THEN
$st(p), pt := WT, (pt + 1) \bmod \mathsf{card}(PR)$
END ;
ent $\mathrel{\widehat{=}}$
ANY p WHERE
$p \in Wtg \wedge pid(p) = pt$
THEN
$st(p) := AC$
END ;
rel $\mathrel{\widehat{=}}$
ANY p WHERE
$p \in Act$
THEN
$st(p), pt := ID, (pt + 1) \bmod \mathsf{card}(PR)$
END ;

The state of each process is recorded in the variable st, which is a total function from the set of processes PR to the set of states ST. A pointer pt is used to select the process which changes its state. The pointer is incremented when a process requests access for the critical section or when a process releases the critical section. The pointer is incremented modulo $\mathsf{card}(PR)$, giving in this way a sequential behavior to the system. Processes are selected in the order determined by the constant function pid, which is a bijection from PR to the interval $0..\mathsf{card}(PR) - 1$.

The safety properties of this system are specified by invariant predicates. The mutual exclusion property is specified as the invariant $\mathsf{card}(Act) \leq 1$, which states that there is at most one process in the critical section. On another hand, the invariant states that the critical section is empty when the pointer selects a process in state *idle* or *waiting*. Finally, the last invariant specifies that any process in the critical section is pointed by pt.

When the pointer selects a process q which is already waiting for the critical section, q will eventually enter into it. This is specified by $q \in Wtg \wedge pid(q) = pt \leadsto q \in Act$, which follows from the BRL basic rule and the property

$$q \in Wtg \wedge pid(q) = pt \gg_m q \in Act$$

In order to verify this *ensures* relation, the MP0 and MP1 rules must be proved [1]:

MP0: $q \in Wtg \wedge pid(q) = pt \Rightarrow [req \,[\!]\, ent \,[\!]\, rel]\; q \in Act$

MP1: $q \in Wtg \wedge pid(q) = pt \Rightarrow (grd(req) \vee grd(ent) \vee grd(rel))$

Calculating the weakest precondition of events *req*, *ent* and *rel*, MP0 follows from the following implications:

$$\forall p \cdot (q \in Wtg \wedge pid(q) = pt \wedge p \in Idl \wedge pid(p) = pt \Rightarrow q \in Act) \tag{3.2}$$

$$\forall p \cdot (q \in Wtg \wedge pid(q) = pt \wedge p \in Wtg \wedge pid(p) = pt \Rightarrow q \in Act \cup \{p\}) \tag{3.3}$$

$$\forall p \cdot (q \in Wtg \wedge pid(q) = pt \wedge p \in Act \Rightarrow q \in Act - \{p\}) \tag{3.4}$$

(3.2) and (3.4) hold trivially because *req* and *rel* are miraculous events in a state satisfying $q \in Wtg \wedge pid(q) = pt$. In fact $pid(q) = pt$ and $pid(p) = pt$ in both premises allow to conclude that $p = q$ (in (3.4), $pid(p) = pt$ follows from $p \in Act$ and the invariant $\forall p \cdot (p \in Act \Rightarrow pid(p) = pt)$). Therefore $q \in Wtg \wedge p \in Idl$ of $q \in Wtg \wedge p \in Act$ are both equivalent to *false* when $p = q$. In the same way, $p = q$ follows from the premise of (3.3), which easily proves its conclusion. Finally, MP1 follows directly from the premise $q \in Wtg \wedge pid(p) = pt$ which implies the guard of event *ent*.

□

In spite of the fact that certain liveness properties can be proved without need for variant functions, the induction rule IND for *leads-to* is still fundamental in the proof of general liveness properties. The following examples show specifications and proofs of liveness properties where the variant functions are widely used.

Example 3.3
In this example, we prove that *MutexSeq*, specified in the example 3.2, satisfies a stronger property: *any process requesting access to the critical section eventually gets access to it:*

$$q \in Wtg \leadsto q \in Act$$

[1] From now on, instead of giving the premise of proof obligations MP0 and MP1 as $I \wedge P \wedge \neg Q$, they are given as P, considering $I \wedge \neg Q$ as assumptions.

So as to prove this property, the induction rule must be applied and a variant function must be defined. The first consideration is to define a function calculating the "distance" between q, the process requesting access, and the pointer pt. Formally, it is defined as follows:

$$V(pt,q) = \begin{cases} 0 & \text{if } pt = pid(q) \\ pid(q) - pt & \text{if } pt < pid(q) \\ \mathsf{card}(PR) - pt + pid(q) & \text{if } pt > pid(q) \end{cases}$$

It can be proved that execution of events *req* and *rel* decrements $V(pt,q)$ while q is waiting for the critical section. However, the distance is not modified by execution of event *ent*; this event only decrements the number of processes in state *waiting*. Therefore, the variant must be a lexicographic one:

$$W(pt,q,st) \mathrel{\widehat{=}} [V(pt,q), \mathsf{card}(st^{-1}[\{WT\}])]$$

Using the variant function W, the following basic liveness property can be proved in *MutexSeq* under a minimal progress assumption:

$$q \in Wtg \wedge W(pid,q,st) = [m,n] \gg_m (q \in Wtg \wedge W(pid,q,st) \prec [m,n]) \vee q \in Act$$

Now, by applying the basic and induction rules for *leads-to*, the property $q \in Wtg \rightsquigarrow q \in Act$ is proved.

Verification of the basic liveness property follows from the MP0 and MP1 rules:

MP0: $q \in Wtg \wedge W(pt,q,st) = [m,n] \Rightarrow$
$[req \,[\!]\, ent \,[\!]\, rel]\ ((q \in Wtg \wedge W(pt,q,st) \prec [m,n]) \vee q \in Act)$

MP1: $q \in Wtg \wedge W(pt,q,st) = [m,n] \Rightarrow (grd(req) \vee grd(rel) \vee grd(rel))$.

Calculating the weakest preconditions of events *req*, *ent* and *rel*, MP0 follows from the following implications:

$$\begin{aligned} & q \in Wtg \wedge W(pt,q,st) = [m,n] \wedge p \in Idl \wedge pid(p) = pt \\ & \quad \Rightarrow ((q \in Wtg \cup \{p\} \wedge W(pt \oplus 1, q, st \lhd\!\!+ \{p \mapsto WT\}) \prec [m,n]) \vee q \in Act) \qquad (3.5) \\ & q \in Wtg \wedge W(pt,q,st) = [m,n] \wedge p \in Wtg \wedge pid(p) = pt \\ & \quad \Rightarrow ((q \in Wtg - \{p\} \wedge W(pt,q,st \lhd\!\!+ \{p \mapsto AC\}) \prec [m,n]) \vee q \in Act \cup \{p\}) \qquad (3.6) \\ & q \in Wtg \wedge W(pt,q,st) = [m,n] \wedge p \in Act \\ & \quad \Rightarrow ((q \in Wtg \wedge W(pt \oplus 1, q, st \lhd\!\!+ \{p \mapsto ID\}) \prec [m,n]) \vee q \in Act - \{p\}) \qquad (3.7) \end{aligned}$$

where $pt \oplus 1$ is $pt+1 \bmod \mathsf{card}(PR)$. (3.5) and (3.7) are proved in a similar way; only the proof of (3.5) is commented. When $pid(q) = pt$, (3.5) holds trivially, because from $pid(p) = pt$ follows that $p = q$ and the premise becomes *false*. On another hand, when $pid(q) \neq pt$, it must be remarked from the definition of V, that $V(pt \oplus 1, q) < V(pt,q)$. Now, from definition of W and $V(pt,q) = m$ follows $V(pt \oplus 1, q) < n$, which proves the decrement of W in the lexicographic order. Moreover, $q \in Wtg$ continues to hold when p is added to Wtg, which gives the proof of (3.5). A similar case analysis allows the proof of (3.6). When $pid(q) = pt$, it follows that $p = q$ and the conclusion $q \in Act \cup \{p\}$ holds. When $pid(q) \neq pt$, $q \in Wtg - \{p\}$ holds and the cardinality of Wtg decreases, which entails the decrement of W in the lexicographic order, considering that pt does not change; this fact gives the proof of

(3.6). WF1 is proved by case analysis as well. When $Act \neq \varnothing$ the guard of *rel* holds. In the contrary case, *pt* selects a process in state *idle* or *waiting*, which implies the guard of events *req* or *ent* respectively. Hence, MP1 holds.

□

The proofs of general liveness properties in the previous examples take a regular pattern :

- identification of a variant function,
- proof of a basic liveness property containing the variant and
- application of the basic (BRL) and induction (IND) rules to derive the goal.

Depending on both the complexity of the system and the property to verify, it is very common to need a variant in a lexicographic order and it may be difficult to find one, which is decremented by any event in the system. This problem is addressed in [34] and briefly described in section 2.4.4. The proposal states that variants must be associated with (new) events instead of variables. In this way, a kind of refinement is proposed where "new events" do not refine *skip*, they are bounded by a variant and do not modify the variants associated with others events in previous refinements. In this way, the verification of modalities proceeds in a stepwise approach. The initial model presents a reduced set of behaviors where a liveness property is proved by the execution of an event. Complementary behaviors are introduced in a series of refinements which terminates when all behaviors are considered. Constraints in the introduction of new events give the proof of the modality. The approach to the specification and proof of liveness in B events systems presented in this thesis, allows modular proofs where case analysis is directly supported by the logic and there is no need to develop a system by the kind of refinements proposed in [34]. The following example illustrates this fact.

Example 3.4
System *MutexSeq*, presented in example 3.2, imposes strong constraints in the specification. The variable *pt* has the role of a scheduler which selects the next process to execute. All processes change their state in a sequential way, defined by its order in the function *pid*. This specification allows simple liveness proofs. Now, a little different system is presented, it enables execution of events *req* among the processes in any interleaved way, but it continues to impose a sequential execution of event *ent* among the processes. This system is named *MutexSeqP*, it contains the same variables, constants, properties and invariants of system *MutexSeq*. Events *ent* and *rel* do not change. Event *req* does not verify the value of the pointer and it does not modify its value. Events of *MutexSeqP* are

$req \mathrel{\widehat{=}}$
ANY p WHERE
$p \in Idl$
THEN
$st(p) := WT$
END ;

$ent \mathrel{\widehat{=}}$
ANY p WHERE
$p \in Wtg \wedge pid(p) = pt$
THEN
$st(p) := AC$
END ;

$rel \mathrel{\widehat{=}}$
ANY p WHERE
$p \in Act$
THEN
$st(p) := ID \parallel$
$pt := (pt + 1) \bmod \mathsf{card}(PR)$
END ;

Any process q waiting for the critical section in system *MutexSeqP* eventually gets access to it. This property, formally specified by

$$q \in Wtg \leadsto q \in Act$$

can be proved in many ways, using the rules for the *leads-to* relation.

A way, similar to the approach taken in [7], is to find a variant which is decremented by all events in the system when q is waiting. It allows specification of the property

$$q \in Wtg \wedge \mathcal{V}(q) = v \gg_m (q \in Wtg \wedge \mathcal{V}(q) \prec v) \vee q \in Act$$

where $\mathcal{V}(q)$ is the lexicographic variant $[V(pt, q), \mathsf{card}(Idl), \mathsf{card}(Wtg)]$, v is the list $[l, m, n]$ with l, m and n universally quantified over $\mathbb{N}$, $\prec$ is the order relation in the lexicographic order and $V(pt, q)$ is the distance between the pointer and $pid(q)$, defined in example 3.3.

The basic property can be proved by the MP0 and MP1 rules. MP0 states that any event decrements the variant while q waits for the critical section or q becomes active. Execution of *req* preserves $q \in Wtg$ and decrements the number of processes in state *idle* while the pointer is not modified, which leads to the decrement of W. When event *ent* is executed, if q is selected, it becomes active. If q is not selected, q remains waiting and the variant is decremented because the number of processes in state *waiting* decrements and the pointer is not modified. Finally, when *rel* is executed, q remains waiting for the critical section and the distance between $pid(q)$ and pt is reduced. These reasonings give the proof of MP0. On another hand, MP1 follows by case analysis. If there is an active process, the guard of *rel* holds. In the contrary case, the pointer selects a process in state *idle* and the guard of *req* holds, or the pointer selects a process in state *waiting* and the guard of *ent* holds. It proves MP1. The main liveness property $q \in Wtg \rightsquigarrow q \in Act$ follows from the basic property by application of BRL and IND rules.

The rules for the *leads-to* relation allow one to make modular proofs structured by case analysis, simplifying the variant functions. The following paragraphs show a proof illustrating this fact.

When q waits for the critical section and the pointer selects it, q eventually becomes active. On another hand, if q is not selected by the pointer, it will eventually be. This reasoning can be formalized by the following properties::

$$q \in Wtg \wedge pid(q) = pt \rightsquigarrow q \in Act \tag{3.8}$$

$$q \in Wtg \wedge pid(q) \neq pt \rightsquigarrow q \in Wtg \wedge pid(q) = pt \tag{3.9}$$

Now, from these properties and the rules TRA and DSJ, follows that q becomes eventually active when it waits for the critical section. The proof is as comes next: The property $q \in Wtg \wedge pid(q) \neq pt \rightsquigarrow q \in Act$ follows from (3.9) and transitivity with (3.8). The goal $q \in Wtg \rightsquigarrow q \in Act$ is proved from this last derivation and (3.8) by the disjunction rule DSJ.

When q is waiting for the critical section and pt selects it, system *MutexSeqP* only can execute event *act* which ensures that q becomes active or it can execute several times event *req*. The number of executions of *req* is bounded by the cardinality of Idl, the set of processes in state *idle*. This reasoning allows specification of the following basic property:

$$\begin{aligned} &q \in Wtg \wedge pid(q) = pt \wedge \mathsf{card}(Idl) = m \gg_m \\ &\quad (q \in Wtg \wedge pid(q) = pt \wedge \mathsf{card}(Idl) < m) \vee q \in Act \end{aligned} \tag{3.10}$$

which proves, by application of BRL rule, the property (3.8). In order to verify that (3.10) holds in *MutexSeqP*, MP0 and MP1 rules must be verified:

$$\begin{aligned} &q \in Wtg \wedge pid(q) = pt \wedge \mathsf{card}(Idl) = m \Rightarrow \\ &\quad [req \,[\!]\, ent \,[\!]\, rel]\ (q \in Wtg \wedge pid(q) = pt \wedge \mathsf{card}(Idl) < m) \vee q \in Act \\ &q \in Wtg \wedge pid(q) = pt \wedge \mathsf{card}(Idl) = m \Rightarrow (grd(req) \vee grd(ent) \vee grd(rel)) \end{aligned}$$

When q is waiting and it is selected by pt, from the invariant follows that event *rel* is not enabled and it establishes any postcondition. Event *req* decrements the variant while q is selected and execution of *act* allows that q becomes active. This proves MP0 and gives arguments to verify MP1.

The proof of (3.9) considers the cases when a process is active or the critical section is empty. These cases are specified by properties (3.11) and (3.12) below. In the first case, property (3.11), event *ent* is not enabled and execution of *rel* reduces the distance between the selected process and q. The execution of *req* is bounded by the cardinality of *Idl* and its execution does not modify the pointer. Therefore, the execution of *rel* eventually leads to a state where q is selected. In the second case, property (3.12), when there is no active process, the event *rel* is not enabled, the execution of *req* is bounded by $\mathsf{card}(Idl)$ and execution of both *req* and *ent* do not modify pt. For that reason, the distances among the processes and the pointer do not change when the critical section is empty and a process eventually enters the critical section. This reasoning allows the proof of the following properties:

$$\mathcal{P} \wedge \neg R \wedge \mathsf{card}(Idl) = m \gg_m (\mathcal{P} \wedge \neg R \wedge \mathsf{card}(Idl) < m) \vee \mathcal{Q} \tag{3.11}$$

$$\mathcal{P} \wedge R \wedge \mathsf{card}(Idl) = m \gg_m (\mathcal{P} \wedge R \wedge \mathsf{card}(Idl) < m) \vee (\mathcal{P} \wedge \neg R) \tag{3.12}$$

where

$$\mathcal{P} \equiv P \wedge V(pt, q) = n \qquad P \equiv q \in Wtg \wedge pid(q) \neq pt$$
$$\mathcal{Q} \equiv (P \wedge V(pt, q) < n) \vee Q \qquad Q \equiv q \in Wtg \wedge pid(q) = pt$$
$$R \equiv Act = \varnothing$$

(3.11) and (3.12) are proved by MP0 and MP1 rules, using the arguments explained above. Finally, property (3.9) is derived as comes next:

1. $\mathcal{P} \wedge \neg R \rightsquigarrow \mathcal{Q}$; (3.11), BRL and IND
2. $\mathcal{P} \wedge R \rightsquigarrow \mathcal{P} \wedge \neg R$; (3.12), BRL and IND
3. $\mathcal{P} \wedge R \rightsquigarrow \mathcal{Q}$; 2, 1 and TRA
4. $\mathcal{P} \rightsquigarrow \mathcal{Q}$; 3, 1 and DSJ
5. $P \wedge V(pt, q) = n \rightsquigarrow (P \wedge V(pt, q) < n) \vee Q$; 4 and def. $\mathcal{P}$ and $\mathcal{Q}$
6. $P \rightsquigarrow Q$; 5 and IND
7. $q \in Wtg \wedge pid(q) \neq pt \rightsquigarrow q \in Wtg \wedge pid(q) = pt$; 7 and def. P and Q

□

3.3 Weak Fairness

3.3.1 Specification of Liveness Properties

A weak fairness assumption in a B event system states that any event, continuously enabled, is infinitely often executed. This supposition guarantees the execution of events making the transitions specified by basic liveness properties. These events are known as *helpful* events and they are explicitly noted in basic liveness properties, which takes the general form $G \cdot P \gg_w Q$, where G is the helpful event; this property must be pronounced as follows: "by event G, P ensures Q".

Informally, the property $G \cdot P \gg_w Q$ holds in a system if execution of any event in a reachable state where P and $\neg Q$ holds, preserves P or establishes Q, G is enabled and

establishes Q. Formally, basic liveness properties under weak fairness assumptions are defined as follows

Definition 3.3 (ensures under weak fairness)
For any B *event system* $\mathcal{S}$ *with set of events* E*, helpful event* G *in* E *and state predicates* P *and* Q*,*

$$G \cdot P \gg_w Q \mathrel{\widehat{=}} SI \wedge P \wedge \neg Q \Rightarrow (([S]\ (P \vee Q)) \wedge grd(G) \wedge [G]\ Q)$$

where S *is the bounded choice* $[]_{e \in E}\ e$*.*

From this definition follows that execution of any event of $\mathcal{S}$ in a reachable state where $P \wedge \neg Q$ holds, terminates in a state where $P \vee Q$ holds. Moreover, it states that the helpful event establishes Q and is enabled while $P \wedge \neg Q$ holds. Therefore, given that G is continuously enabled when Q is not *true*, by the weak fairness assumption, G will be eventually executed and Q will be *true*.

As in definition 3.2, the calculation of SI makes difficult the practical application of definition 3.3 to verify basic liveness properties. Therefore, the invariant I of the event system $\mathcal{S}$ is used to propose the following rules which are sufficient to prove basic liveness properties under weak fairness assumptions:

	ANTECEDENT	CONSEQUENT
WF0	$I \wedge P \wedge \neg Q \Rightarrow [S]\,(P \vee Q)$	$G \cdot P \gg_w Q$
WF1	$I \wedge P \wedge \neg Q \Rightarrow grd(G) \wedge [G]\,Q$	

where S, as in definition 3.3, stands for the bounded choice of events $[]_{e \in E}\ e$ and G is a helpful event in E. WFO and WF1 proof obligations are sufficient to prove basic liveness properties because $SI \Rightarrow I$ and therefore, by definition 3.3, the basic property follows from these rules.

Proof obligations of basic liveness properties under weak fairness assumptions are simpler than the ones for minimal progress. The proof of a property $P \gg Q$ under minimal progress requires that any event establishes Q, while under minimal progress, only the helpful event establishes Q and the rest maintains P or establishes Q. This fact, allows simple variants in proofs by induction, as it can be seen in the following examples.

Example 3.5
The example 3.4 shows the proof of the property $q \in Wtg \rightsquigarrow q \in Act$ under a minimal progress assumption in the system *MutexSeqP*. The proof follows from the following properties:

$$q \in Wtg \wedge pid(q) = pt \rightsquigarrow q \in Act \tag{3.8}$$
$$q \in Wtg \wedge pid(q) \neq pt \rightsquigarrow q \in Wtg \wedge pid(q) = pt \tag{3.9}$$

In this example, (3.8) and (3.9) are proved by basic liveness properties under the weak fairness assumption.

Property (3.8) was proved by the basic property (3.10) under minimal progress. As can be checked, that proof uses the cardinality of *Idl* to bound the execution of event *req* and then ensures that execution of *ent* establishes the goal. Under a weak fairness assumption, the variant is not needed in this case. (3.8) follows simply by application of the BRL rule to

$$ent \cdot q \in Wtg \wedge pid(q) = pt \gg_w q \in Act$$

According to WF0 and WF1 proof obligations, this property is derived from the following implications which are easily proved:

$$q \in Wtg \wedge pid(q) = pt \Rightarrow [req \,[\!]\, ent \,[\!]\, rel]\ (q \in Wtg \wedge pid(q) = pt) \vee q \in Act$$
$$q \in Wtg \wedge pid(q) = pt \Rightarrow ([ent]\ q \in Act) \wedge grd(ent)$$

In example 3.4, property (3.9) was proved by basic properties (3.11) and (3.12). Under weak fairness assumptions (3.9) follows from the following properties:

$$req \cdot \mathcal{P} \wedge \neg grd(ent) \wedge \mathsf{card}(Idl) = m \gg_w (\mathcal{P} \wedge \neg grd(ent) \wedge \mathsf{card}(Idl) < m) \vee (\mathcal{P} \wedge grd(ent)) \quad (3.13)$$
$$ent \cdot P \wedge V(pt,q) = n \wedge Act = \varnothing \wedge grd(ent) \gg_w P \wedge V(pt,q) = n \wedge Act \neq \varnothing \quad (3.14)$$
$$rel \cdot P \wedge V(pt,q) = n \wedge Act \neq \varnothing \gg_w (P \wedge V(pt,q) < n) \vee Q \quad (3.15)$$

where

$$\mathcal{P} \equiv P \wedge V(pt,q) = n \wedge Act = \varnothing \qquad P \equiv q \in Wtg \wedge pid(q) \neq pt$$
$$Q \equiv q \in Wtg \wedge pid(q) = pt$$

The basic properties are verified by the WF0 and WF1 proof obligations. These proofs follow from the arguments used in the verification of properties (3.11) and (3.12) in the example 3.4. The proof of (3.9) under weak fairness assumptions is as follows:

1. $\mathcal{P} \wedge \neg grd(ent) \rightsquigarrow \mathcal{P} \wedge grd(ent)$; (3.13) and IND
2. $\mathcal{P} \wedge grd(ent) \rightsquigarrow P \wedge V(pt,q) = n \wedge Act \neq \varnothing$; (3.14) and IND
3. $\mathcal{P} \wedge \neg grd(ent) \rightsquigarrow P \wedge V(pt,q) = n \wedge Act \neq \varnothing$; 1, 2 and TRA
4. $P \wedge V(pt,q) = n \wedge Act = \varnothing \rightsquigarrow P \wedge V(pt,q) = n \wedge Act \neq \varnothing$; 3, 2, and DSJ def. $\mathcal{P}$
5. $P \wedge V(pt,q) = n \wedge Act \neq \varnothing \rightsquigarrow (P \wedge V(pt,q) < n) \vee Q$; (3.15) and IND
6. $P \wedge V(pt,q) = n \wedge Act = \varnothing \rightsquigarrow (P \wedge V(pt,q) < n) \vee Q$; 4, 5 and TRA
7. $P \wedge V(pt,q) = n \rightsquigarrow (P \wedge V(pt,q) < n) \vee Q$; 5, 6 and DSJ
8. $q \in Wtg \wedge pid(q) \neq pt \rightsquigarrow q \in Wtg \wedge pid(q) = pt$; 7, IND and def. P, Q

□

Example 3.6
This example presents a classical solution to the mutual exclusion problem. The solution is given by an abstract model where processes waiting for the critical section are served in first input first output basis. The model considers a log where waiting processes are recorded with a number denoting their time of request. The log is represented by a partial injection lg from the set of processes PR to $\mathbb{N}$. If the log is empty, a requesting process is recorded in lg with any natural number. If the log is not empty, the requesting time of a process must be greater than the maximum of the times recorded in lg. The requesting process with the lowest requesting time can access the critical section if it is empty. When a process enters the critical section, its record is removed from lg. The main safety property of this system states that there is at most one process in the critical section. It is stated as the invariant $\mathsf{card}(Act) \leq 1$ in *Mutex* system. The auxiliary invariant $\mathsf{dom}(lg) = Wtg$ states that all waiting process are recorded in the log. The invariant $\forall p \cdot (p \in Wtg \Rightarrow \exists n \cdot (n \in \mathbb{N} \wedge lg(p) - \mathsf{min}(\mathsf{ran}(lg)) = n))$ states that the waiting period of any waiting process is a natural number. This invariant is needed in the liveness proof. The specification of system *Mutex* is as follows:

SYSTEM	EVENTS
$\quad Mutex$	$req \mathrel{\widehat{=}}$
SETS	$\quad$ ANY p, t WHERE
$\quad PR;$	$\quad\quad p \in Idl \wedge t \in \mathbb{N} \wedge (lg \neq \varnothing \Rightarrow t > \mathsf{max}(\mathsf{ran}(lg)))$
$\quad ST = \{ID, WT, AC\}$	$\quad$ THEN
PROPERTIES	$\quad\quad st(p), lg := WT, lg \cup \{p \mapsto t\}$
$\quad \mathsf{card}(PR) > 1$	$\quad$ END ;
DEFINITIONS	$ent \mathrel{\widehat{=}}$
$\quad Idl \mathrel{\widehat{=}} st^{-1}[\{ID\}]$;	$\quad$ ANY p WHERE
$\quad Wtg \mathrel{\widehat{=}} st^{-1}[\{WT\}]$;	$\quad\quad p \in Wtg \wedge Act = \varnothing \wedge lg(p) = \mathsf{min}(\mathsf{ran}(lg))$
$\quad Act \mathrel{\widehat{=}} st^{-1}[\{AC\}]$;	$\quad$ THEN
VARIABLES	$\quad\quad st(p), lg := AC, \{p\} \lhd lg$
$\quad st, lg$	$\quad$ END ;
INVARIANT	$rel \mathrel{\widehat{=}}$
$\quad st \in PR \rightarrow ST \wedge$	$\quad$ ANY p WHERE
$\quad lg \in PR \rightarrowtail\!\!\!\!\!+ \mathbb{N} \wedge$	$\quad\quad p \in Act$
$\quad \mathsf{dom}(lg) = Wtg \wedge$	$\quad$ THEN
$\quad \mathsf{card}(Act) \leq 1$	$\quad\quad st(p) := ID$
INITIALISATION	$\quad$ END ;
$\quad st, lg := PR \times \{ID\}, \varnothing$	

The main liveness property of *Mutex* states that any requesting process eventually gets access to the critical section: $q \in Wtg \leadsto q \in Act$. The proof uses the following predicates:

$$O \equiv Act = \varnothing \qquad P \equiv q \in Wtg$$
$$Q \equiv q \in Act \qquad R \equiv lg(q) = \mathsf{minran}(lg))$$

The proof needs a variant which calculates the time that q has to wait until it can access the critical section:

$$V = lg(q) - \mathsf{min}(\mathsf{ran}(lg))$$

The liveness property is derived from the following lemmas:

$$P \wedge O \wedge R \leadsto Q \tag{3.16}$$
$$P \wedge O \wedge \neg R \wedge V = n \leadsto (P \wedge \neg O \wedge R) \vee (P \wedge \neg O \wedge \neg R \wedge V < n) \vee Q \tag{3.17}$$
$$P \wedge \neg O \wedge R \leadsto P \wedge O \wedge R \tag{3.18}$$
$$P \wedge \neg O \wedge \neg R \wedge V = m \leadsto P \wedge O \wedge \neg R \wedge V = m \tag{3.19}$$

(3.16) states that q eventually enters the critical section when it is the oldest waiting process and there is no process in state *active*. (3.17) specifies that the variant is decremented when q is not the oldest process in the log or the request of q eventually becomes the oldest. (3.18) and (3.19) express that neither R nor V is modified when the critical section is released. The

proof of $q \in Wtg \leadsto q \in Act$ is as follows:

1.	$P \wedge \neg O \wedge R \leadsto Q$	; (3.18), (3.16) and TRA
2.	$P \wedge O \wedge \neg R \wedge V = n \leadsto (P \wedge \neg O \wedge \neg R \wedge V < n) \vee Q$	; (3.17), 1 and CAN
3.	$P \wedge O \wedge \neg R \wedge V = n \leadsto (P \wedge O \wedge \neg R \wedge V < n) \vee Q$	; 2, (3.19) and CAN
4.	$P \wedge O \wedge \neg R \leadsto Q$	; 3 and IND
5.	$P \wedge \neg O \wedge \neg R \wedge V = n \leadsto Q$	; (3.19), 4 and TRA
6.	$P \Rightarrow \exists n \cdot (n \in \mathbb{N} \wedge V = n)$	; invariant
7.	$P \wedge \neg O \wedge \neg R \leadsto Q$	; 5, DSJ, 6 and replacement
8.	$P \wedge \neg O \leadsto Q$	; 7, 1 and TRA
9.	$P \wedge \neg(O \wedge R) \leadsto Q$	; 4, 8 and DSJ
10.	$q \in Wtg \leadsto q \in Act$	; 9, (3.16) and DSJ

Lemmas (3.16)–(3.19) are derived from the following basic properties:

$$ent \cdot q \in Wtg \wedge Act = \varnothing \wedge \mathsf{min}(\mathsf{ran}(lg)) = k \wedge lg(q) = l \tag{3.20}$$
$$\gg_w (q \in Wtg \wedge Act \neq \varnothing \wedge \mathsf{min}(\mathsf{ran}(lg)) > k \wedge lg(q) = l) \vee q \in Act$$
$$rel \cdot q \in Wtg \wedge Act \neq \varnothing \wedge \mathsf{min}(\mathsf{ran}(lg)) = k \wedge lg(q) = l \tag{3.21}$$
$$\gg_w q \in Wtg \wedge Act = \varnothing \wedge \mathsf{min}(\mathsf{ran}(lg)) = k \wedge lg(q) = l$$

The proof of lemmas (3.16)–(3.19) uses the predicates O, P, Q and R defined above. Moreover, $m(lg)$ denotes $\mathsf{min}(\mathsf{ran}(lg))$

Proof of (3.16)

(3.20)
$\Rightarrow$ { BRL }
$P \wedge O \wedge m(lg) = k \wedge lg(q) = l \leadsto (P \wedge \neg O \wedge m(lg) > k \wedge lg(q) = l) \vee Q$
$\Rightarrow$ { $k = l$ is a boolean constant expression and SCJ }
$P \wedge O \wedge m(lg) = k \wedge lg(q) = l \wedge k = l \leadsto (P \wedge \neg O \wedge m(lg) > k \wedge lg(q) = l \wedge k = l) \vee Q$
$\Rightarrow$ { $m(lg) > lg(q) \equiv false$ }
$P \wedge O \wedge m(lg) = k \wedge lg(q) = l \wedge k = l \leadsto Q$
$\Rightarrow$ { DSJ over k and l }
$\exists(k, l) \cdot (P \wedge O \wedge m(lg) = k \wedge lg(q) = l \wedge k = l) \leadsto Q$
$\Rightarrow$ { one point rule and def. R }
$P \wedge O \wedge R \leadsto Q$

Proof of (3.17)

Let $\mathcal{P} \equiv P \wedge O \wedge m(lg) = k \wedge lg(q) = l$ and $\mathcal{Q} \equiv P \wedge \neg O \wedge m(lg) > k \wedge lg(q) = l$ in the

following proofs:

(3.20)
$\Rightarrow$ { BRL and def. $\mathcal{P}$ and $\mathcal{Q}$ }
$\mathcal{P} \leadsto \mathcal{Q} \vee Q$
$\Rightarrow$ { SCJ and weakening the right hand side }
$\mathcal{P} \wedge l - k = n \wedge k < l \leadsto (\mathcal{Q} \wedge l - k = n \wedge k < l) \vee Q$
$\Rightarrow$ { $\mathcal{Q} \wedge l - k = n \wedge k < l \Rightarrow P \wedge \neg O \wedge (m(lg) = lg(q) \vee (m(lg) \neq lg(q) \wedge lg(q) - m(lg) < n))$, R and V }
$\mathcal{P} \wedge l - k = n \wedge k < l \leadsto (P \wedge \neg O \wedge R) \vee (P \wedge \neg O \wedge \neg R \wedge V < n) \vee Q$
$\Rightarrow$ { DSJ over k and l }
$\exists(k, l) \cdot (\mathcal{P} \wedge l - k = n \wedge k < l) \leadsto (P \wedge \neg O \wedge R) \vee (P \wedge \neg O \wedge \neg R \wedge V < n) \vee Q$
$\equiv$ { def. $\mathcal{P}$ and one point rule }
$P \wedge O \wedge lg(q) - m(lg) = n \wedge m(lg) < lg(q) \leadsto (P \wedge \neg O \wedge R) \vee (P \wedge \neg O \wedge \neg R \wedge V < n) \vee Q$
$\equiv$ { $\neg(m(lg) > lg(q))$, def. R and V }
$P \wedge O \wedge \neg R \wedge V = n \leadsto (P \wedge \neg O \wedge R) \vee (P \wedge \neg O \wedge \neg R \wedge V < n) \vee Q$

Proof of (3.18)

(3.21)
$\Rightarrow$ { BRL }
$P \wedge \neg O \wedge m(lg) = k \wedge lg(q) = l \leadsto P \wedge O \wedge m(lg) = k \wedge lg(q) = l$
$\Rightarrow$ { SCJ }
$P \wedge \neg O \wedge m(lg) = k \wedge lg(q) = l \wedge k = l \leadsto P \wedge O \wedge m(lg) = k \wedge lg(q) = l \wedge k = l$
$\Rightarrow$ { weakening RHS and def. R }
$P \wedge \neg O \wedge m(lg) = k \wedge lg(q) = l \wedge k = l \leadsto P \wedge O \wedge R$
$\Rightarrow$ { DSJ over k and l }
$\exists(k, l) \cdot (P \wedge \neg O \wedge m(lg) = k \wedge lg(q) = l) \leadsto P \wedge O \wedge R$
$\equiv$ { one point rule and def. R }
$P \wedge \neg O \wedge R \leadsto P \wedge O \wedge R$

Proof of (3.19)

(3.21)
$\Rightarrow$ { BRL }
$P \wedge \neg O \wedge m(lg) = k \wedge lg(q) = l \leadsto P \wedge O \wedge m(lg) = k \wedge lg(q) = l$
$\Rightarrow$ { SCJ }
$P \wedge \neg O \wedge m(lg) = k \wedge lg(q) = l \wedge k \neq l \wedge l - k = m \leadsto P \wedge O \wedge m(lg) = k \wedge lg(q) = l \wedge k \neq l \wedge l - k = m$
$\Rightarrow$ { weakening RHS, def. R and V }
$P \wedge \neg O \wedge m(lg) = k \wedge lg(q) = l \wedge k \neq l \wedge l - k = m \leadsto P \wedge O \wedge \neg R \wedge V = m$
$\Rightarrow$ { disjunction over k and l }
$\exists(k, l) \cdot (P \wedge \neg O \wedge m(lg) = k \wedge lg(q) = l \wedge k \neq l \wedge l - k = m) \leadsto P \wedge O \wedge \neg R \wedge V = m$
$\equiv$ { one point rule, def. R and V }
$P \wedge \neg O \wedge \neg R \wedge V = m \leadsto P \wedge O \wedge \neg R \wedge V = m$

Finally, the basic properties (3.20) and (3.21) are proved by the WF0 and WF1 rules. □

3.3.2 Bounded Nondeterminism

Events specified by the ANY construct, introducing a bounded nondeterministic choice, can be considered as a family of deterministic events. Therefore, an event e specified by

$$e \mathrel{\widehat{=}} \text{ANY } z \text{ WHERE } P_z \text{ THEN } S_z \text{ END}$$

where P_z implies that z belongs to the set of values $z_1, \ldots, z_n$, for a certain value n, can be considered equal to the bounded choice of deterministic events:

$$e = e_1 \,[\!]\, \ldots \,[\!]\, e_n$$

where

$$e_i \mathrel{\widehat{=}} \text{SELECT } P_i \text{ THEN } S_i \text{ END} \quad \text{for } i \in z_1 \ldots z_n$$

This equality is justified by the equality of generalized substitutions defined in [2], where two substitutions are equal if and only if they have the same weakest precondition:

$$[e]\, R \equiv [e_1 \,[\!]\, \ldots \,[\!]\, e_n]\, R$$

for any postcondition R.

Let $\mathcal{S}$ be a system including e and $\mathcal{S}'$ a similar system which distinguishes itself from $\mathcal{S}$ by including the deterministic events $e_1 \ldots e_n$ instead of e. The equality between e and the bounded choice $e_1 \,[\!]\, \ldots \,[\!]\, e_n$ guarantees that $\mathcal{S}$ and $\mathcal{S}'$ satisfy the same invariants (safety properties). On another hand, $\mathcal{S}$ and $\mathcal{S}'$ satisfy the same liveness properties under minimal progress assumptions. This assertion comes from the fact that MP0 and MP1 proof obligations for basic properties are equivalent for $\mathcal{S}$ and $\mathcal{S}'$. Unfortunately, under weak fairness assumptions, the liveness properties of $\mathcal{S}$ and $\mathcal{S}'$ are not the same. Any basic property $e_i \cdot P(z_i) \gg_w Q(z_i)$ in $\mathcal{S}'$ proved by WF0 and WF1 proof obligations, cannot be satisfied by system $\mathcal{S}$. In fact, WF0 is satisfied by $\mathcal{S}$, but WF1 cannot be proved. In other words, $\mathcal{S}$ cannot guarantee that event e establishes $Q(z_i)$ when it is executed in any reachable state satisfying $P(z_i) \wedge \neg Q(z_i)$.

In this thesis, bounded nondeterministic events like e are considered as a syntactical shorthand for the family of events $e_1, \ldots, e_n$. It allows for using any deterministic event e_i as a helpful event in the proof of basic liveness properties under weak fairness assumptions. Moreover, it is advisable to reduce the nondeterminism of e only at a certain level by considering a particular condition $R(z)$. That is, e can be seen as shorthand for the choice $e_1 \,[\!]\, e_2$ where:

$$e_1 \mathrel{\widehat{=}} \text{ANY } z \text{ WHERE } P_z \wedge R(z) \text{ THEN } S_z \text{ END}$$
$$e_2 \mathrel{\widehat{=}} \text{ANY } z \text{ WHERE } P_z \wedge \neg R(z) \text{ THEN } S_z \text{ END}$$

These considerations simplify the proof of basic liveness properties under weak fairness assumptions, as it is illustrated in the following example.

Example 3.7
The example 3.1 shows that system PC, specified in example 2.15, satisfies property $d \in P \rightsquigarrow d \in C$ under a minimal progress assumption. Under a weak fairness assumption the proof is very similar; it follows from the induction (IND) and basic (BRL) rules applied to:

$$pc \cdot d \in P \wedge \mathsf{card}(P) = n \gg_w d \in P \wedge \mathsf{card}(P) < n \vee d \in C$$

Nondeterminism in event pc requires a variant function to guarantee that any data d will eventually be transferred from P to C. Considering D as a finite set and taking pc as a shorthand for events pc_1 and pc_2 defined as comes next:

$pc_1 \mathrel{\widehat{=}}$
ANY d' WHERE
$d' \in P \wedge d' = d$
THEN
$C, P := C \cup \{d'\}, P - \{d'\}$
END ;

$pc_2 \mathrel{\widehat{=}}$
ANY d' WHERE
$d' \in P \wedge d' \neq d$
THEN
$C, P := C \cup \{d'\}, P - \{d'\}$
END

the property $d \in P \rightsquigarrow d \in C$ follows directly from

$$pc_1 \cdot d \in P \gg_w d \in C$$

by application of the BRL rule. WF0 and WF1 allows the proof of this basic property:

WF0: $d \in P \Rightarrow [pc_1 \,[\!]\, pc_2 \,[\!]\, env]\, d \in P \vee d \in C$.

WF1: $d \in P \Rightarrow ([pc_1]\, d \in C) \wedge grd(pc_1)$

WF0 is easily proved considering that $pc = pc_1 \,[\!]\, pc_2$. WF1 follows from the calculations of the guard and weakest precondition of pc_1: $[pc_1]\, d \in C \equiv d \in C \cup \{d\}$ and $grd(pc_1) \equiv d \in P$. □

3.4 Preserving Liveness Properties

The implementation of abstract B models is made by stepwise refinement, as indicated in section 2.4.3 of chapter 2. The classical approach to prove correctness of a refinement step is the verification of five conditions:

1. Initialization. The initialization statement must establish the concrete invariant.
2. Refined events. Concrete events must refine the abstract ones.
3. New events. New events must refine *skip*.
4. Bounded skip. New events must decrement a variant.
5. Deadlock-freeness. The refinement does not introduce more deadlocks than the abstraction.

Condition 1–3 allows preservation of safety properties, while conditions 4 and 5 guarantees that modalities, specified under minimal progress assumptions, are preserved.

The main concern of this section is the proposal of proof rules for preserving general liveness properties in the refinement of abstract models. Considering that general liveness properties are derived from basic properties, it suffices to prove the preservation of basic liveness properties. Therefore, the approach taken in this thesis to prove the correctness of a refinement step is to guarantee that safety properties are preserved, by conditions 1–3 of the classical approach, and then verify that basic liveness properties are preserved. However, the conditions to preserve basic liveness properties depend on the fairness assumption taken

in the specification of the basic properties. For this reason, the conditions to preserve these properties are grouped in proof rules for minimal progress and weak fairness assumptions respectively.

We suppose that a certain property $P \gg Q$ holds in an abstract model $\mathcal{S}$. This model has a state variable x, invariant I and set of events E. The bounded choice of events in E is denoted by S. We consider a refinement of $\mathcal{S}$ denoted by $\mathcal{T}$, with state variable y and gluing invariant J. The set of events in $\mathcal{T}$ is made up of concrete events, refining the abstract ones, and new events refining *skip*. The bounded choice of concrete events is denoted by T ($S \sqsubseteq_J T$) and the bounded choice of new events is denoted by H ($skip \sqsubseteq H$). In the following sections, we propose sufficient conditions to prove that the basic property $P \gg Q$, proved in $\mathcal{S}$ under minimal progress or weak fairness assumptions, is preserved in the refinement $\mathcal{T}$.

3.4.1 Preservation under Minimal Progress

The proof of the basic property $P \gg_m Q$ in $\mathcal{S}$ needs the verification of rules MP0 and MP1:

MP0 $I \wedge P \wedge \neg Q \Rightarrow [S]\, Q$.

MP1 $I \wedge P \wedge \neg Q \Rightarrow grd(S)$.

These rules guarantee that in any state of $\mathcal{S}$ where $P \wedge \neg Q$ holds, an event is enabled and its execution establish Q. In order to guarantee the preservation of $P \gg_m Q$ in $\mathcal{T}$, the refined system must guarantee that any concrete event establishes Q' when it is executed in any state where $P' \wedge \neg Q'$ holds, where P' and Q' are predicates on the refined state where there exists an abstract state satisfying both the gluing invariant and P or Q respectively:

$$P' \equiv \exists x \cdot (I(x) \wedge J(y,x) \wedge P)$$
$$Q' \equiv \exists x \cdot (I(x) \wedge J(y,x) \wedge Q)$$

Considering that concrete events T refine S, and by the very definition of refinement, the establishment of Q' by T can be asserted; for this reason, the MP0 rule continues to hold in $\mathcal{T}$. On another hand, as a result of refinement, the guard of T becomes stronger than the guard of S. Therefore, rule MP1 does not hold in $\mathcal{T}$, and the execution of concrete events cannot be ensured.

In order to guarantee execution of concrete events, the "bounded skip" and "deadlock-freeness" conditions of the classical approach to verify refinements are used:

BMP	$I \wedge J \wedge V = n \Rightarrow [H]\, V \prec n$
LMP	$I \wedge J \wedge grd(S) \Rightarrow grd(T) \vee grd(H))$

In fact, the refinement of *skip* by new events guarantees that the predicate $\exists x \cdot (I(x) \wedge J(y,x) \wedge P \wedge \neg Q)$ is invariant under H. This fact guarantees that P' holds while Q' is not established. However, it does not allow necessary the execution of concrete events because new events can take control forever. The rule BMP prevents this anomalous behavior by demanding that new events decrement a variant over a well founded set. Moreover, the LMP rule guarantees that some event in the refinement $\mathcal{T}$ is enabled whenever any abstract event is enabled. In this way, when $P' \wedge \neg Q'$ holds in the refinement $\mathcal{T}$, the execution of any concrete event eventually reaches a state where Q' holds. The following examples show the application of these rules to verify the preservation of liveness properties under minimal progress assumptions.

Example 3.8
In example 3.1, we prove that the producer-consumer system PC, specified in the example 2.15, satisfies under the minimal progress assumption the property $d \in P \rightsquigarrow d \in C$, stating that any data d in the producer set, is eventually transferred to the consumer set. The liveness property is derived from the basic liveness property

$$d \in P \wedge \mathsf{card}(P) = n \gg_m (d \in P \wedge \mathsf{card}(P) < n) \vee d \in C$$

In example 2.17, the refined producer-consumer system *PC1* is presented. This refinement introduces a new variable, *buf*, which models a buffer between the producer and consumer, and a new event *cons*. The gluing invariant $C = C_c \cup buf$ relates the abstract consumer set with the concrete one and the buffer. For the sake of simplicity, the events of PC and *PC1* are presented again:

$pc \mathrel{\widehat{=}}$	$env \mathrel{\widehat{=}}$
ANY d WHERE	SELECT
$d \in P$	$P = \varnothing$
THEN	THEN
$P, C := P - \{d\}, C \cup \{d\}$	$P, C, := D, \varnothing$
END ;	END

Events of system PC

$prod \mathrel{\widehat{=}}$	$cons \mathrel{\widehat{=}}$	$env_c \mathrel{\widehat{=}}$
ANY d WHERE	ANY d WHERE	SELECT
$d \in P$	$d \in buf$	$P = \varnothing \wedge buf = \varnothing$
THEN	THEN	THEN
$P := P - \{d\} \parallel$	$C_c := C_c \cup \{d\} \parallel$	$P, C_c, buf := D, \varnothing, \varnothing$
$buf := buf \cup \{d\}$	$buf := buf - \{d\}$	END
END ;	END ;	

Events of system *PC1*

According to BMP and LMP rules, the abstract property $p \in P \rightsquigarrow p \in C$ is preserved in the refinement *PC1* if the new event *cons* decrements a variant and $grd(pc \ [\!] \ env) \Rightarrow grd(prod \ [\!] \ cons \ [\!] \ env)$. Considering that execution of the event *cons* removes elements from *buf*, the obvious variant in this case is the cardinality of *buf*. Now it is easy to prove the following implication:

$$\mathsf{card}(buf) = n \wedge d \in buf \Rightarrow \mathsf{card}(buf - \{d\}) < n$$

which proves the BMP rule. On another hand, if the abstract event *pc* is enabled, the producer set is not empty, therefore the concrete event *prod* is enabled. When the producer set is empty in the abstraction, two cases must be considered: the buffer in the refined system is empty or not. When the producer set and the buffer are both empty, the guard of the event env_c, which models the environment in the refined system, is enabled. However, if the buffer is not empty, the guard of event *cons*, which models the consumer, is enabled. This reasoning can be concluded by the following implications, which allow the proof of the LMP condition:

$$grd(pc) \Rightarrow grd(prod)$$
$$grd(env) \wedge buf = \varnothing \Rightarrow grd(env_c)$$
$$grd(env) \wedge buf \neq \varnothing \Rightarrow grd(cons)$$

□

Example 3.9
This example presents *MutexSeqPT*, a refinement of system *MutexSeqP*, specified in the example 3.4. Moreover, it gives the proof of the preservation of the liveness property $q \in Wtg \rightsquigarrow q \in Act$ which holds in *MutexSeqP*.

The refinement changes the pointer pt of *MutexSeqP*, which selects the process that access the critical section, by a token. As in *MutexSeqP*, processes access the critical section in the order defined by their *pid*. The variable tk, a total function from PR to the booleans, records which process has assigned the token. Initially, the token is assigned to the process with *pid* zero. There is always a process with the token ($\mathsf{card}(tk^{-1}[\{true\}]) = 1$). A requesting process p enters the critical section when the token is assigned to it ($p \in st^{-1}[\{WT\}] \cap tk^{-1}[\{true\}]$); therefore, any process in state *active* has the token ($st^{-1}[\{AC\}] \subseteq tk^{-1}[\{true\}]$). The token is assigned by a server which uses a mailbox to communicate with the processes. The mailbox is modeled by a sequence of processes ch of size equal or lower than one. When a process p leaves the critical section, p is stored in the mailbox. Any process in the mailbox is in state *idle* and has assigned the token ($\mathsf{ran}(ch) \subseteq st^{-1}[\{ID\}] \cap tk^{-1}[\{true\}]$). The gluing invariant considers two cases. First, when the mailbox is not empty, the pointer pt is the identifier of the process in the mailbox plus one, modulo $\mathsf{card}(PR)$ ($(ch \neq \varnothing \Rightarrow pt = (pid(ch(1))+1) \bmod \mathsf{card}(PR))$). Second, when the mailbox is empty, the pointer corresponds to the identifier of the process which holds the token ($\forall p \cdot (p \in tk^{-1}[\{true\}] \wedge ch = \varnothing \Rightarrow pt = idx(p))$). The complete specification of the refinement *MutexSeqPT* is as follows:

REFINEMENT
MutexSeqPT

REFINES
MutexSeqP

VARIABLES
st, ch, tk

INVARIANT
$ch \in \mathsf{seq}(PR) \wedge$
$tk \in PR \rightarrow BOOL \wedge$
$\mathsf{size}(ch) \leq 1 \wedge$
$\mathsf{card}(tk^{-1}[\{true\}]) = 1 \wedge$
$(ch \neq \varnothing \Rightarrow pt =$
$(pid(ch(1)) + 1) \bmod \mathsf{card}(PR)) \wedge$
$\forall p \cdot (p \in tk^{-1}[\{true\}] \wedge ch = \varnothing \Rightarrow$
$pt = pid(p)) \wedge$
$\mathsf{ran}(ch) \subseteq st^{-1}[\{ID\}] \cap tk^{-1}[\{true\}] \wedge$
$st^{-1}[\{AC\}] \subseteq tk^{-1}[\{true\}]$

INITIALISATION
$st := PR \times \{ID\} \parallel$
$ch := \varnothing \parallel$
$tk := (PR \times false) \lhd\!\!+ \{pid^{-1}(0) \mapsto true\}$

EVENTS
req $\widehat{=}$
ANY p WHERE
$p \in st^{-1}[\{ID\}] \wedge ch = \varnothing$
THEN
$st(p) := WT$
END ;
ent $\widehat{=}$
ANY p WHERE
$p \in st^{-1}[\{WT\}] \cap tk^{-1}[\{true\}]$
THEN
$st(p) := AC$
END ;
rel $\widehat{=}$
ANY p WHERE
$p \in st^{-1}[\{AC\}]$
THEN
$st(p), ch := ID, ch \leftarrow p$
END ;
srv $\widehat{=}$
SELECT
$ch \neq \varnothing$
THEN
$ch := \varnothing \parallel tk := tk \lhd\!\!+ \{ch(1) \mapsto false,$
$pid^{-1}((pid(ch(1)) + 1) \bmod \mathsf{card}(PR))$
$\mapsto true\}$
END ;

Processes in state *idle* must verify that the mailbox is empty before changing its state to *waiting*. If the mailbox is not empty, the server has not yet assigned the token to the next process, and a process in state *idle* can access the critical section again. The server is modeled by a new event named *srv*, which is enabled when the mailbox is not empty. When *srv* is executed, the mailbox is cleared and the token is assigned to the next processes in the order stated by *pid*.

Apart from safety properties specified as invariants and commented above, this system satisfies the liveness property $q \in Wtg \rightsquigarrow q \in Act$, which is preserved from the abstraction. In order to demonstrate this fact, the BMP and LMP rules must be proved. The new event *srv* is enabled when the mailbox *ch* is not empty, and its execution removes an element from *ch*. Therefore, the variant decremented by *srv* in this case is the cardinality of *ch*. The following implication, which follows from the weakest precondition of *srv* proves the BMP rule:

$$\mathsf{card}(ch) = n \wedge ch \neq \varnothing \Rightarrow \mathsf{card}(ch) < n$$

When the mailbox is not empty, it means that a process has left the critical section and the guard on *srv* becomes enabled. Let req_c, act_c and rel_c be the guards of *req*, *act* and *rel* in the refinement respectively. If the mailbox is empty, and there is a process in state *idle* (the guard of *req* in *MutexSeqP* holds), then the guard of req_c is enabled. When a waiting process is pointed by *pt* in the abstraction (the guard of *act* in *MutexSeqP* is enabled) there is no direct information about the token. If the token is assigned to a process in state *idle*, then the guard of req_c is enabled when the mailbox is empty. If the token is assigned to an active process, then the guard of rel_c is enabled. Finally, if a waiting process has assigned the token, then the guard of act_c holds. This reasoning allows the proof of the following formulas:

$$grd(req \,[\!]\, ent \,[\!]\, rel) \wedge ch \neq \varnothing \Rightarrow grd(srv)$$
$$grd(req) \wedge ch = \varnothing \Rightarrow grd(req_c)$$
$$grd(act) \wedge tk^{-1}[\{true\}] \subseteq Idl \wedge ch = \varnothing \Rightarrow grd(req_c)$$
$$grd(act) \wedge tk^{-1}[\{true\}] \subseteq Act \Rightarrow grd(rel_c)$$
$$grd(act) \wedge tk^{-1}[\{true\}] \subseteq Wtg \Rightarrow grd(act_c)$$
$$grd(rel) \Rightarrow grd(rel_c)$$

It must be remarked that $ch = \varnothing$ follows from $grd(rel)$ or $tk^{-1}[\{true\}] \subseteq \overline{Idl}$ under the assumptions of the invariant. These implications allows the proof of the LMP rule. Finally, the proofs of BMP and LMP rules guarantee that $q \in Wtg \rightsquigarrow q \in Act$ holds in the refinement. □

3.4.2 Preservation under Weak Fairness

A basic liveness property $G \cdot P \gg_w Q$ holds in the model $\mathcal{S}$ under a weak fairness assumption, if the WF0 and WF1 rules hold:

WF0 $P \wedge \neg Q \Rightarrow [S]\,(P \vee Q)$

WF1 $P \wedge \neg Q \Rightarrow ([G]\,Q) \wedge grd(G)$

These rules guarantee that execution of any event preserves P or establishes Q. Moreover, the helpful event is enabled and its execution establishes Q. The weak fairness assumption ensures that G will be eventually executed.

The choice of abstract events S can be considered as $F \,[\!]\, G$, where G is a helpful event in E, the set of events in $\mathcal{S}$, and F is the bounded choice of events in $E - \{G\}$. In the refined system $\mathcal{T}$, the concrete events become F' and G', where F' is the refinement of F and G' the refinement of G. New events refining *skip* are denoted by H. By the very definition of refinement, the following predicates hold in $\mathcal{T}$:

$$I \wedge J \wedge P \wedge \neg Q \Rightarrow [F' \,[\!]\, H]\, \exists x \cdot (I \wedge J \wedge (P \vee Q))$$
$$I \wedge J \wedge P \wedge \neg Q \Rightarrow [G']\, \exists x \cdot (I \wedge J \wedge Q)$$

However the implication $I \wedge J \wedge P \wedge \neg Q \Rightarrow grd(G')$ does not hold under the same assumptions, because the refined event G' has a guard stronger than $grd(G)$. Then, in order to guarantee the preservation of the basic property, refinement $\mathcal{T}$ must reach a state in the guard of G' when it arrives in a state out of the guard (rule LIP: Liveness Preservation), and that the guard of G' must be preserved by F' and H (rule SAP: Safety Preservation). Formally the proof obligations in $\mathcal{T}$ to preserve $G \cdot P \gg Q$ are:

LIP	$I \wedge J \wedge P \wedge \neg Q \wedge \neg grd(G') \rightsquigarrow grd(G')$
SAP	$I \wedge J \wedge P \wedge \neg Q \wedge grd(G') \Rightarrow [F' \,[\!]\, H]\, grd(G')$

These proof obligations must be proved in all refinements of the abstract system, for each basic property, required in the derivation a *leads-to* property. However, in certain special cases, the basic property can be preserved trivially, or LIP and SAP can be proved by preservation of basic liveness properties, from which LIP follows. These two conditions are stated as two rules which can be applied in the proof of preservation of liveness. In these rules, the predicate $\mathcal{J}_i$, for $i \in \mathbb{N}_1$, denotes the conjunction of gluing invariants:

$$\mathcal{J}_i \equiv J_1 \wedge J_2 \wedge \ldots \wedge J_i \tag{3.22}$$

where J_j, for $j \in 1..i$, denotes the gluing invariant in refinement j. The rules are:

R1.- Let $G_i \cdot P \gg Q$ be a property of the abstract model, or a property in a refinement i, derived from WF0 and WF1 proof obligations, and G_{i+1} the helpful event of the property in a refinement $i+1$. If the following implication holds in that refinement:

$$I \wedge \mathcal{J}_{i+1} \wedge P \wedge \neg Q \Rightarrow grd(G_{i+1})$$

then, the basic property is preserved trivially.

R2.- Let i be a refinement where the LIP and SAP proof obligations hold:

$$I \wedge \mathcal{J}_i \wedge P \wedge \neg Q \wedge \neg grd(G_i) \rightsquigarrow grd(G_i)$$
$$I \wedge \mathcal{J}_i \wedge P \wedge \neg Q \wedge grd(G_i) \Rightarrow [e]\, grd(G_i)$$

where e is universally quantified over the set $E_i - \{G_i\}$, and E_i is the set of events in refinement i. Let β be the set of basic liveness properties, sufficient to derive the LIP proof obligation in refinement i. If the following implication holds in refinement $i+1$:

$$I \wedge \mathcal{J}_i \wedge J_{i+1} \wedge grd(G_i) \Rightarrow grd(G_{i+1})$$

where J_{i+1} is the gluing invariant in refinement $i+1$, and if all basic properties in β are preserved in refinement $i+1$, then LIP and SAP proof obligations hold in refinement $i+1$.

Application of rule R1 happens when the guard of the helpful event is not modified, or it is modified but, it is still implied by the gluing invariant. Rule R2 is applied when the helpful event is not modified any more, or its guard becomes weaker, and the refinement process proceeds with other events in the system.

The following examples show the application of the R1 rule in the proof of preservation of basic liveness properties. The application of R2 rule is presented in a larger example of chapter 6. Soundness of R1 and R2 rules is proved in appendix E.

Example 3.10
In example 3.7 we prove that the producer-consumer system PC, specified in the example 2.15, satisfies the property $d \in P \rightsquigarrow d \in C$ under a weak fairness assumption, stating that any data d in the producer set, is eventually transferred to the consumer set. The proof is given in two forms. The first one uses a nondeterministic helpful event and a variant. The second form does not need a variant and the helpful event is deterministic. In this example we prove that the liveness property is preserved in the refinement *PC1*, presented in example 2.17, under the weak fairness assumption. Events in both the specification (PC) and the refinement (*PC1*), can be seen in page 59.

The first form of the proof uses the event *pc* to guarantee the basic property in PC

$$pc \cdot d \in P \wedge \mathsf{card}(P) = n \gg_w d \in P \wedge \mathsf{card}(P) < n \vee d \in C$$

This property is trivially preserved in *PC1*, considering that the guard of *pc* in the abstraction and the guard of *prod* in the refinement, are the same. Therefore, the implication $d \in P \Rightarrow grd(prod)$ holds and, by the R1 rule, the basic properitty is preserved in *PC1*.

The second form of the proof uses the deterministic event *pc1* to prove the property

$$pc_1 \cdot d \in P \gg_w d \in C$$

In the refinement *PC1*, a corresponding refined event $prod_1$ can be obtained by considering the equality $prod = prod_1 \,[\!]\, prod_2$, where $prod_1$ and $prod_2$ are defined as follows:

$prod_1 \mathrel{\widehat{=}}$
ANY d' WHERE
$d' \in P \wedge d' = d$
THEN
$buf, P := buf \cup \{d'\}, P - \{d'\}$
END ;

$prod_2 \mathrel{\widehat{=}}$
ANY d' WHERE
$d' \in P \wedge d' \neq d$
THEN
$buf, P := buf \cup \{d'\}, P - \{d'\}$
END ;

It must be remarked that $prod_1$ refines pc_1. Therefore, the basic property is trivially preserved in *PC1* by the R1 rule, because the implication $d \in P \Rightarrow grd(prod_1)$ holds.

We note that in this example, and in example 3.8, the preservation of the general liveness property $d \in P \rightsquigarrow d \in C$ implies that the refined system *PC1* must satisfy the property:

$$d \in P \rightsquigarrow d \in C_c \cup buf$$

However, this property does not guarantee that any data d in P will be eventually in the concrete store C_c. In order to guarantee this transfert, the following property must be proved:

$$d \in buf \rightsquigarrow d \in C_c$$

Taking the considerations of the section 3.3.2, and remarking the equality of events $cons = cons_1 \,[\!]\, cons_2$, where $cons_1$ and $cons_2$ are as follows:

$cons_1 \mathrel{\widehat{=}}$
ANY d' WHERE
$d' = d \wedge d' \in buf$
THEN
$C_c := C_c \cup \{d'\} \parallel$
$buf := buf - \{d'\}$
END ;

$cons_2 \mathrel{\widehat{=}}$
ANY d' WHERE
$d' \neq d \wedge d' \in buf$
THEN
$C_c := C_c \cup \{d'\} \parallel$
$buf := buf - \{d'\}$
END

the following basic property holds in system *PC1*:

$$cons_1 \cdot d \in buf \gg_w d \in C_c$$

From this basic property, and the preserved general liveness property, the property

$$d \in P \rightsquigarrow d \in C_c$$

is derived as follows:

1.	$cons_1 \cdot d \in buf \gg_w d \in C_c$	; basic property
2.	$d \in P \rightsquigarrow d \in C_c \cup buf$	; preserved property
3.	$d \in buf \rightsquigarrow d \in C_c$	; 1 and BRL
4.	$d \in P \rightsquigarrow d \in C_c \vee d \in buf$	; 2
5.	$d \in P \rightsquigarrow d \in C_c$	; 4, 3 and CAN

□

Example 3.11
In this example we present a refinement of system *Mutex* presented in example 3.6. This refinement, named *Mutex2*, implements the log *lg* of *Mutex* by an injective sequence *qu* where requesting processes are served in a first input first output basis. *qu* is initialized to the empty sequence; the range of *qu* corresponds to the set of waiting processes ($Wtg = \mathsf{ran}(qu)$). A new event *srv* is introduced, modeling a server which assigns a token to the first process in *qu*. The token is represented by a boolean; the variable *tk*, a total function from *PR* to *BOOL*, records the process which has assigned the token. Initially, for any process p, $tk(p)$ is *false*. At most, there is a process which has assigned the token ($\mathsf{card}(Tk) \leq 1$). If a waiting process has assigned the token, then it is the first process in *qu* ($Wtg \cap Tk \subseteq \{\mathsf{first}(qu)\}$). Any process in the critical section has assigned the token ($Act \subseteq Tk$); therefore the token is not assigned to processes in state *idle* ($Tk \subseteq PR - Idl$). The gluing invariant relating the log *lg* with the sequence *qu* states that for any waiting process p, its position $qu^{-1}(p)$ is $lg(p) - \mathsf{min}(\mathsf{ran}(lg)) + 1$. The refined system is as comes next:

REFINEMENT
Mutex2
REFINES
Mutex
DEFINITIONS
$Idl \mathrel{\widehat{=}} st^{-1}[\{ID\}]$;
$Wtg \mathrel{\widehat{=}} st^{-1}[\{WT\}]$;
$Act \mathrel{\widehat{=}} st^{-1}[\{AC\}]$;
$Tk \mathrel{\widehat{=}} tk^{-1}[\{true\}]$;
CONSTANTS
sh
PROPERTIES
$sh = \lambda l \cdot (l \in ST \mathrel{+\!\!\!\rightarrow} \mathbb{N} \mid$
$\lambda x \cdot (x \in \mathbb{N} \wedge x \in \mathsf{ran}(l) \mid$
$(x - \mathsf{min}(\mathsf{ran}(l))) + 1))$
VARIABLES
st, qu, tk
INVARIANT
$qu \in \mathsf{iseq}(PR) \wedge$
$tk \in PR \rightarrow BOOL \wedge$
$Wtg = \mathsf{ran}(qu) \wedge$
$\mathsf{card}(Tk) \leq 1 \wedge$
$Wtg \cap Tk \subseteq \{\mathsf{first}(qu)\} \wedge$
$Act \subseteq Tk \wedge$
$Tk \subseteq PR - Idl \wedge$
$(lg \mathbin{;} sh(lg))^{-1} = qu \wedge$
INITIALISATION
$st := PR \times \{ID\} \parallel$
$qu := \varnothing \parallel$
$tk := (PR \times \{false\})$

EVENTS
$req \mathrel{\widehat{=}}$
ANY p WHERE
$p \in Idl$
THEN
$st(p) := WT \parallel$
$qu := qu \leftarrow p$
END ;

$ent \mathrel{\widehat{=}}$
ANY p WHERE
$p \in Wtg \cap Tk$
THEN
$st(p) := AC \parallel$
$qu := \mathsf{tail}(qu)$
END ;

$rel \mathrel{\widehat{=}}$
ANY p WHERE
$p \in Act$
THEN
$st(p) := ID \parallel$
$tk(p) := false$
END ;

$srv \mathrel{\widehat{=}}$
SELECT
$qu \neq \varnothing \wedge Tk = \varnothing$
THEN
$tk(\mathsf{first}(qu)) := true$
END

Apart from safety properties stated as invariants, *Mutex2* preserves the liveness property $q \in Wtg \rightsquigarrow q \in Act$ specified in the abstraction *Mutex*. This property is derived from the following basic properties holding in *Mutex*:

$$ent{\cdot}q \in Wtg \wedge Act = \varnothing \wedge \mathsf{min}(\mathsf{ran}(lg)) = k \wedge lg(q) = l \gg_w (q \in Wtg \wedge Act \neq \varnothing \wedge \mathsf{min}(\mathsf{ran}(lg)) > k \wedge lg(q) = l) \vee q \in Act \tag{3.20}$$

$$rel{\cdot}q \in Wtg \wedge Act \neq \varnothing \wedge \mathsf{min}(\mathsf{ran}(lg)) = k \wedge lg(q) = l \gg_w q \in Wtg \wedge Act = \varnothing \wedge \mathsf{min}(\mathsf{ran}(lg)) = k \wedge lg(q) = l \tag{3.21}$$

In order to guarantee the preservation of $q \in Wtg \rightsquigarrow q \in Act$, (3.20) and (3.21) must be preserved in *Mutex2*.

The guard of *rel*, the helpful event of (3.21) in both the abstraction and the refinement, is the same. Therefore, the implication $Act \neq \varnothing \Rightarrow grd(rel)$ holds, and by the R1 rule, (3.21) is trivially preserved.

Considering that the guard of *act*, the helpful event in (3.20), is equivalent to ($Wtg \cap Tk \neq \varnothing$) and replacing G' with the helpful event *act* in LIP, and $F' \,[\!]\, H$ with $req \,[\!]\, rel \,[\!]\, srv$ in the

SAP rule, the sufficient conditions to guarantee the preservation of (3.20) are:

$$q \in Wtg \wedge Wtg \cap Tk = \varnothing \rightsquigarrow Wtg \cap Tk \neq \varnothing \tag{3.23}$$

$$q \in Wtg \wedge Wtg \cap Tk \neq \varnothing \Rightarrow [req \,[\!]\, rel \,[\!]\, srv]\ Wtg \cap Tk \neq \varnothing \tag{3.24}$$

When the token is not assigned to a waiting process, and q is still waiting for the critical section, it means that the set of processes holding the token is empty ($Tk = \varnothing$) and that the sequence qu is not empty. Therefore the guard of event *srv* is enabled and its execution assigns the token to a waiting process, enabling the guard of event *act*. This reasoning and the WF0 and WF1 proof obligations guarantee that the following basic property holds:

$$srv \cdot q \in Wtg \wedge Wtg \cap Tk = \varnothing \gg_w Wtg \cap Tk \neq \varnothing$$

Then, the proof obligation (3.23) follows from this basic property applying the BRL rule.

The proof of (3.24) can be argued as follows. When the guard of *act* is established, that is, the token has been assigned to a waiting process, the execution of event *req* does not modify this condition. On another hand, under this same condition, the events *rel* and *srv* are not enabled. In fact, when the token is assigned to a waiting process, it means that the critical section is empty ($Act = \varnothing$), and that the set of processes holding the token is not empty ($Tk \neq \varnothing$). Therefore, when the guard of *act* is enabled, the events *rel* and *srv* are miraculous and they establish any postcondition.

□

3.5 Conclusion

In this chapter we present the main practical aspects of our approach. We illustrate by examples how to specify and prove liveness properties in abstract models and then how to preserve them in refinements. We explain how to derive general liveness properties from basic liveness properties by application of three rules without fairness considerations.

Basic liveness properties are defined considering two fairness assumptions: minimal progress and weak fairness; these definitions consider the strongest invariant. Then we propose the proof obligations MP0 and MP1 which are sufficient conditions to verify a basic properties $P \gg_m Q$ under minimal progress assumptions. Under the assumption of the invariant, these rules state that all events establish Q when they are executed in a state satisfying $P \wedge \neg Q$ (MP0) and that at least, one event is enable (MP1). The verification of a basic properties $G \cdot P \gg_w Q$, where G denotes the helpful event of the property, under weak fairness assumptions, is done by the rules WF0 and WF1. These rules state that under the assumption of the invariant, the execution of any event in a state where $P \wedge \neg Q$ terminates in a state where P or Q holds (WF0) and that the helpful event is enabled and its execution establishes Q (WF1). Events defined by ANY constructs introducing bounded nondeterminism, are considered as a finite family of deterministic events which can be used as helpful events in basic properties under weak fairness assumptions.

Preservation of liveness properties depends on fairness assumptions. We state that in order to preserve general liveness properties, it suffices to preserve the basic ones. In the case of minimal progress, the rules BMP and LMP, which do not depend on a particular basic property, are proposed. The rule BMP states that any new event in a refinement must decrement a variant function, and the LMP rule indicates that under the assumption of the gluing invariant and the invariant of the abstraction, the guards of concrete and new events

must be implied by the guards of the abstraction. In the case of weak fairness, the proposed rules LIP and SAP are sufficient conditions to preserve a basic property $G \cdot P \gg_w Q$. The LIP rule states that we need to guarantee that the refined system reaches a state where the guard of G', the refined helpful event, holds whenever it arrives in a state where $\neg grd(G')$ holds. The SAP rule indicates that the guard of G' must be preserved by all events in the refinement. Considering that the proof of the LIP rule depends on another basic properties, these properties must be preserved in the refinement steps. We proposed rules R1 and R2 which are used to provide trivial proofs in the preservation of basic properties under weak fairness assumptions.

In [56] we proposed the first time our approach to the specification and proof of liveness properties in B event systems in a UNITY-like style considering minimal progress and weak fairness assumptions. The proof obligation WF1 was less general that the version presented in this thesis: it was only valid for deterministic events. In [60] we reviewed our rules for the verification of basic liveness properties under weak fairness assumptions and we proposed the rules for their preservation under refinement. The approach was illustrated by the specification and refinement of a mutual exclusion algorithm similar to system *Mutex2* in example 3.11. In [17] we presented the rules for verification and preservation of basic properties under minimal progress assumptions and we applied these rules in the liveness proofs of the Ricart-Agrawala mutual exclusion algorithm.

Chapter 4

A Semantics for *leads-to*

Résumé

Dans ce chapitre, nous donnons les justifications théoriques de notre travail. Nous présentons un cadre ensembliste théorique, où les événements d'un système et son itération sont modélisés comme des transformateurs d'ensembles. Afin de donner une interprétation aux propriétés de type leads-to*, nous introduisons la relation de* terminaison*. D'autre part, la relation* leads-to *est présentée sous la forme d'une relation entre sous-ensembles d'états, nommée relation d'*atteignabilité*, qui est définie d'une manière inductive. Ces définitions sont données sans aucune hypothèse d'équité dans l'itération des événements. Nous montrons, par un théorème, que ces relations sont équivalentes. Ensuite, les relations de terminaison et d'atteignabilité sont instanciées afin de considérer les hypothèses d'équité de progrès minimal ou d'équité faible. Nous prouvons que ces instanciations vérifient toujours l'équivalence entre les relations de terminaison et d'atteignabilité. Nous donnons des exemples de dérivation de règles permettant la preuve des propriétés de vivacité en utilisant notre analyse sémantique.*

In action or event systems, such as UNITY, TLA or actions systems, the behavior of a system is described in terms of observations about its state, and they are known as *state based* formalisms. All of these formalisms have a common aspect: their semantics is founded on state-traces of transition systems.

State traces of transition systems impose an operational reasoning about the behavior of a system. However, this operational behavior can be hidden by using temporal logic to specify safety and liveness properties. Semantics of temporal formulas is given by state-traces of transition systems. A proof system allows derivation of properties from other proved properties without an operational reasoning, only by symbolic calculations. Soundness and completeness of the logic are established by proofs relating logical formulas with assertions about state-traces. So, at this point we come back to operational reasoning about the transition systems.

A more abstract possibility to define the semantics of action systems is to found it on fixpoints of set (or predicates) transformers. Inspired from [7] and [37], we characterize certain liveness properties as fixpoints of set transformers modeling iteration of events, under minimal progress or weak fairness assumptions. We are only interested in properties of type P *leads-to* Q, where P and Q are predicates on the state space of a system, with the informal meaning:

"the system eventually *reaches* a state satisfying Q when its execution arrives at any state in P". The fixpoint characterizing this property denotes the largest subset of states, containing all states satisfying P, where iteration of events in the system, is guaranteed to *terminate* in a state satisfying Q. In this chapter, soundness and completeness of rules allowing derivation of *leads-to* properties are proved by demonstrating that notions of reachability and termination are equal under minimal progress or weak fairness assumptions. Aside from soundness and completeness, two examples of applications of these results are presented. The first one is a proof of sufficient conditions for liveness properties under minimal progress given in [7]. The second one is an original result which give sufficient conditions to derive a liveness property under minimal progress when the given property holds under weak fairness.

The chapter is structured as follows. In Section 4.1, the mathematical notation of set transformer is introduced. Set transformers are used to model the weakest or liberal weakest precondition of events in a set theoretical framework. In Section 4.2, the semantics is developed and equality (soundness and completeness) between notions of termination and reachability is proved. A formal link between the fixpoint semantics and a state-traces semantics is presented in the subsection 4.2.2. The semantics is instantiated to minimal progress or weak fairness assumptions in subsections 4.2.3 and 4.2.4. In Section 4.3 we prove soundness of the rules for basic liveness properties. Finally, Section 4.4 presents examples of sufficient conditions to derive liveness properties using the results of the previous section.

4.1 Set Transformers

This section is divided in two parts. The first part presents an event system as a set transformer and gives the notion of liberal set transformer. The second part presents the dovetail operator that is used to model a weak fairness assumption.

4.1.1 Set Transformers

A set transformer is a total function of type $\mathbb{P}(\mathcal{U}) \rightarrow \mathbb{P}(\mathcal{U})$ for a certain set $\mathcal{U}$. In [2], each generalized substitution F has associated a set transformer $\mathsf{str}(F)$ of type $\mathbb{P}(u) \rightarrow \mathbb{P}(u)$, where u is the state space of a model or refinement with state variable x and invariant I:

$$u = \{z \mid I(z)\}$$

For any r in $\mathbb{P}(u)$, $\mathsf{str}(F)(r)$ denotes the largest subset of states satisfying the weakest precondition of F, that is, the set of states where the execution of F must begin in order for the substitution F to terminate in a state belonging to r. In [7], the events of a B system are formalized by conjunctive set transformers, but instead of identifying the set transformer associated with an event F by $\mathsf{str}(F)$, it is denoted by its name F. In this way $F(r)$ denotes the set $\mathsf{str}(F)(r)$, where F is an event of a B system and r a subset of the state space u. In what follows, this notation is used in this thesis.

Events in a system are modeled by conjunctive set transformers E_i of type $\mathbb{P}(u) \rightarrow \mathbb{P}(u)$, where i belongs to a given finite index set L. Consequently, the system is modeled by a conjunctive set transformer S which is the bounded choice of events E_i:

$$S = []_{i \in L}\ E_i$$

The set of events in a system is denoted by E:

$$E = \{E_i \mid i \in L\}$$

In order to deal with the notion of the weakest liberal precondition of an event F ($wlp(F, \_)$), the *liberal set transformer* of an event F, $\mathcal{L}(F)$, is defined:

Definition 4.1 *The Liberal Set Transformer*

$$\mathcal{L}(F) = \lambda r \cdot (r \in \mathbb{P}(u) \mid \{x \mid x \in u \wedge wlp(F, x \in r)\})$$

The set $\mathcal{L}(F)(r)$ denotes the largest subset of states where the execution of event F must begin in order for F to terminate in a state belonging to r or loop. The liberal set transformer of primitive operators are defined as follows:

$$\begin{array}{ll} \mathcal{L}(skip)(r) = r & \mathcal{L}(F \,;\, G)(r) = \mathcal{L}(F)(\mathcal{L}(G)(r)) \\ \mathcal{L}(F \,[\!]\, G)(r) = \mathcal{L}(F)(r) \cap \mathcal{L}(G)(r) & \mathcal{L}(p \Longrightarrow F)(r) = \overline{p} \cup \mathcal{L}(F)(r) \end{array}$$

where r and p are subsets of u and $\overline{p}$ is $u - p$. In order to satisfy the healthiness condition $\mathcal{L}(F)(u) = u$ of conjunctive set transformers, the liberal set transformer of the preconditioned event is defined by cases:

$$\mathcal{L}(p \mid F)(r) = \begin{cases} p \cap \mathcal{L}(F)(r) & \text{if } r \neq u \\ \mathcal{L}(F)(r) & \text{if } r = u \end{cases}$$

Definitions of liberal set transformers presented here are the set counterpart of definitions in [26].

Aside from the liberal set transformer, we use the set transformers denoting the weakest precondition of classical operators as defined in [7]:

$$\begin{array}{ll} skip(r) = r & (p \Longrightarrow F)(r) = \overline{p} \cup F(r) \\ (F \,[\!]\, G)(r) = F(r) \cap G(r) & (p \mid F)(r) = p \cap F(r) \\ (F \,;\, G)(r) = F(G(r)) & \end{array}$$

The set transformers $F(r)$ and $\mathcal{L}(F)(r)$ for event F and postcondition r are related by the pairing condition:

$$F(r) = \mathcal{L}(F)(r) \cap \mathsf{pre}(F) \tag{4.1}$$

where $\mathsf{pre}(F)$, the termination set of F, is equal to $F(u)$. From the pairing condition follows:

$$F(u) = u \Rightarrow F(r) = \mathcal{L}(F)(r)$$

A set transformer F is *strict* when it respects the excluded miracle law:

$$F(\varnothing) = \varnothing$$

For any set transformer F, when $F(r)$ or $\mathcal{L}(F)(r)$ are recursively defined:

$$F(r) = \mathcal{F}(F(r)) \quad \text{or} \quad \mathcal{L}(F)(r) = \mathcal{G}(\mathcal{L}(F)(r))$$

for monotonic functions $\mathcal{F}$ and $\mathcal{G}$, according to [30], $F(r)$ is taken as the strongest solution of the equation $X = \mathcal{F}(X)$ and $\mathcal{L}(F)(r)$ as the weakest solution of the equation $X = \mathcal{G}(X)$. Considering that these solutions are fixpoints, $F(r)$ is taken as the least fixpoint of $\mathcal{F}$ ($\mathsf{fix}(\mathcal{F})$) and $\mathcal{L}(F)(r)$ as the greatest fixpoint of $\mathcal{G}$ ($\mathsf{FIX}(\mathcal{G})$).

4.1.2 The Dovetail Operator

To model a weak fairness assumption, we use the dovetail operator $\triangledown$ [18], which is a fair nondeterministic choice operator. The dovetail operator is used to model the notion of fair scheduling of two activities. Let A and B be these activities, then the operational meaning of the construct $A \triangledown B$ denotes the execution of commands A and B fairly in parallel, on separate copies of the state, accepting as an outcome any proper, nonlooping, outcome of either A or B. The fair execution of A and B means that neither computation is permanently neglected if favor of the other.

A motivating example of the use of the dovetail operator is given in [18]. In that example the recursive definition:

$$X = (n := 0 \triangledown (X \,;\, n := n + 1))$$

which has as solution "set n to any natural number", is contrasted with the recursion $Y = (n := 0 \,[\!]\, (Y \,;\, n := n + 1))$ which has as solution "set n to any natural number or loop". The possibility of loop in X is excluded with the dovetail operator because the fair choice of statement $n := 0$ will certainly occur. In Y the execution of that statement is not ensured.

The semantic definition for dovetail operator in [18] is given by definition of its weakest liberal precondition predicate transformer (wlp) and its termination predicate hlt. An equivalent definition using the weakest liberal set transformer $\mathcal{L}$ and its termination set pre is given:

Definition 4.2 *The Dovetail Operator*

$$\begin{aligned}
&\mathcal{L}(F \triangledown G)(r) = \mathcal{L}(F)(r) \cap \mathcal{L}(G)(r) \\
&\mathit{pre}(F \triangledown G) = (F(u) \cup G(u)) \cap (\overline{F(\varnothing)} \cup G(u)) \cap (\overline{G(\varnothing)} \cup F(u)) \\
&\mathit{pre}(F \triangledown G) = (F(u) \cap G(u)) \cup (\overline{F(\varnothing)} \cap F(u)) \cup (\overline{G(\varnothing)} \cap G(u))
\end{aligned}$$

The two definitions of the termination set $\mathsf{pre}(F \triangledown G)$ are equivalent; it can be proved by distribution of union over intersection. On another hand, we recall that $\mathsf{grd}(F) = \overline{F(\varnothing)}$.

The set transformer $(F \triangledown G)(r)$, for any r in $\mathbb{P}(u)$ associated with the dovetail operator is obtained from the pairing condition (4.1):

$$(F \triangledown G)(r) = \mathcal{L}(F \triangledown G)(r) \cap \mathsf{pre}(F \triangledown G) \tag{4.2}$$

It must be noted that as far as the liberal set transformer is concerned, the dovetail operator is equal to the choice operator. It differs by having a more liberal pairing condition: to ensures that $F \triangledown G$ halts, it suffices to forbid F and G from both looping and to forbid either from looping in a state where the other fails.

As in [18], we prove that the guard of $F \triangledown G$ is equivalent to the disjunction of the guards of F and G. In terms of sets transformers, the guard of $F \triangledown G$ is stated as follows:

$$\overline{(F \triangledown G)(\varnothing)} = \overline{F(\varnothing)} \cup \overline{G(\varnothing)} \tag{4.3}$$

The proof of this equality is shorter than [18]:

$$
\begin{array}{ll}
& \overline{F(\varnothing)} \cup \overline{G(\varnothing)} \\
= & \\
& \overline{F(\varnothing) \cap G(\varnothing)} \\
= & \qquad \{ \text{ Pairing Condition } \} \\
& \overline{\mathcal{L}(F)(\varnothing) \cap \mathcal{L}(G)(\varnothing) \cap F(u) \cap G(u)} \\
= & \qquad \{ \ F(\varnothing) \cap \overline{F(\varnothing)} = \varnothing \ \} \\
& \overline{\mathcal{L}(F)(\varnothing) \cap \mathcal{L}(G)(\varnothing) \cap (F(u) \cap G(u) \cup F(\varnothing) \cap \overline{F(\varnothing)})} \\
= & \qquad \{ \ \mathcal{L}(F)(\varnothing) \cap F(\varnothing) = \mathcal{L}(F)(\varnothing) \cap F(u) \text{ See note below } \} \\
& \overline{\mathcal{L}(F)(\varnothing) \cap \mathcal{L}(G)(\varnothing) \cap (F(u) \cap G(u) \cup F(u) \cap \overline{F(\varnothing)})} \\
= & \qquad \{ \text{ Similar to two last steps } \} \\
& \overline{\mathcal{L}(F)(\varnothing) \cap \mathcal{L}(G)(\varnothing) \cap (F(u) \cap G(u) \cup F(u) \cap \overline{F(\varnothing)} \cup G(u) \cap \overline{G(\varnothing)})} \\
= & \qquad \{ \text{ Definition of } (F \triangledown G)(\varnothing) \text{ (4.2) and (4.2) } \} \\
& \overline{(F \triangledown G)(\varnothing)}
\end{array}
$$

□

Note In [18] this step requires the proof of $wlp.F.false \Rightarrow grd.F = \neg hlt.F$. We denote this implication as a set expression: $\mathcal{L}(F)(\varnothing) \cap F(\varnothing) = \mathcal{L}(F)(\varnothing) \cap F(u)$. However the proof of this expression is easily given by the pairing condition. We finally note that the sets $F(\varnothing)$ and $F(u)$ are not equal as we can think from the given equality; only the intersection of these sets with $\mathcal{L}(F)(\varnothing)$ is equal.

The dovetail operator is in general non monotonic for the approximation order in commands as defined in [18]. Therefore the existence of least fixed points of recursive equations cannot be proved generally. However, the existence of least fixed points in a restricted class of recursive definitions containing the dovetail operator, is proved in [18]. In this report we only use the dovetail operator to model fair iteration of events. We do not propose the use of this operator to model or refine B event systems. The set transformer modeling fair iteration of events with the dovetail operator is monotonic in the set inclusion order.

4.2 Reachability and Termination

This section proves soundness and (relative) completeness of rules BRL, TRA and DSJ for general liveness properties under minimal progress and weak fairness assumptions in event systems. These rules are sound if for any property $P \leadsto Q$, iteration of events, under minimal progress or weak fairness assumptions, starting in a state satisfying P, leads to a state in the system where Q holds. Completeness of these rules is proved by showing that $P \leadsto Q$ can be derived from the fact that any iteration of events, starting in a state where P holds, terminates into a state satisfying Q.

We do not expect that any iteration of events in a system terminates into a state where the guards of every event are disabled. However, an always terminating iteration can be modeled by supposing, just for the reasoning, that events in the system are embedded in a certain guarded command which models the iteration under a fairness assumption. The iteration only proceeds when the guard of that command is enabled. *Termination* of the iteration will be in a state where the guard does not hold. In this way, if the guard of the iteration is $\neg Q$, and the iteration starts in a state where P holds, the system reaches a state

where Q holds. *Reachability* from P to Q is then associated to termination of the iteration of events. In the following subsection, we formalize our claims in a general framework without concerns of fairness, and then these results are particularized to minimal progress or weak fairness assumptions in other two subsections.

To simplify matters, the strongest invariant [62] is not considered in definitions of this section. In appendix B we restate the results given in this section to consider the strongest invariant.

4.2.1 A General Framework

In this subsection we define a set transformer to model iteration of events and we state its main characteristics. We use this set transformer to define the *termination* relation. Then we give a representation of *leads to* relation in UNITY logic as a relation between subsets of u and we use it to define the *reachability* relation. Finally we prove that the *termination* and the *reachability* relations are equal.

Termination

Let W be a set transformer which models a *step* of the iteration of events S in a system. At this time, the meaning of such a step cannot be defined, however, two properties of W are stated: it must be *monotonic* and *strict*. When the step will be instantiated to a particular fairness assumption, the meaning of W will be given in terms of S. For any r in $\mathbb{P}(u)$, $W(r)$ denotes the largest subset of states where the execution of W must begin in order for W to terminate in a state belonging to r.

To model the iteration of events until the system reaches a state in a certain set r in $\mathbb{P}(u)$, the guarded event $\mathcal{F}(r)$ is defined:

$$\mathcal{F}(r) = (\overline{r} \Longrightarrow W) \tag{4.4}$$

for any $r \in \mathbb{P}(u)$, which allows iteration of W when the system stays in any state in $\overline{r}$. Iteration of $\mathcal{F}(r)$ is modeled by the *opening* operator $\mathcal{F}(r)^{\frown}$. As this operator has a recursive definition:

$$\mathcal{F}(r)^{\frown} = (\mathcal{F}(r)\ ;\ \mathcal{F}(r)^{\frown})\ [\!]\ \mathit{skip}$$

the set where termination of $\mathcal{F}(r)^{\frown}$ is guaranteed ($\mathsf{pre}(\mathcal{F}(r)^{\frown})$) is given by $\mathsf{fix}(\mathcal{F}(r))$ [2].

As W may model an unbounded non determinist set transformer, we use the *Generalized Limit Theorem* to formally justify that any iteration of $\mathcal{F}(r)$ starting in $\mathsf{pre}(\mathcal{F}(r)^{\frown})$ terminates in some state of r. This theorem characterizes the least fixpoint of monotonic functions as an infinite join. We use the version presented in [50], particularizing the theorem to monotonic set transformers. The theorem is as follows:

Theorem 1 (Generalized Limit Theorem)
Let f be a monotonic set transformer, and let f^{α}, for ordinal α, be defined inductively by

$$f^{\alpha} = \bigcup \beta \cdot (\beta < \alpha \mid f(f^{\beta})) \tag{4.5}$$

Then $\mathsf{fix}(f) = f^{\alpha}$ for some ordinal α.

The proof of this theorem is given in [50]. It states that we can choose any ordinal γ, such that $\gamma > \mathsf{card}(\mathsf{dom}(f))$, and then we must have $f^\alpha = f^\beta$ for some $\alpha < \beta < \gamma$. Then it is proved that the common value of f^α and f^β is the least fixpoint of f.

As W is a monotonic function, $\mathcal{F}(r)$ is monotonic, and theorem 1 can be applied to calculate the least fixpoint of $\mathcal{F}(r)$. According to the theorem, it follows that $\mathcal{F}(r)^0 = \varnothing$ and $\mathcal{F}(r)^1 = r$ *because* W is strict. Moreover, for any ordinal α, $\mathcal{F}(r)^{\alpha+1} = \mathcal{F}(r)(\mathcal{F}(r)^\alpha)$ and $\mathcal{F}(r)^\alpha \subseteq \mathcal{F}(r)^{\alpha+1}$. This fact formally supports our claim that the termination set of $\mathcal{F}(r)^\frown$, contains states where any iteration of $\mathcal{F}(r)$ terminates in a state into r. Now, the *termination* relation $\mathcal{T}$ can be defined as follows:

Definition 4.3 (Termination Relation)

$$\mathcal{T} = \{\, a \mapsto b \mid a \subseteq u \wedge b \subseteq u \wedge a \subseteq \mathit{fix}(\mathcal{F}(b)) \,\} \tag{4.6}$$

Reachability

As presented in section 3.1.2, the *leads-to* relation of UNITY logic is defined as a relation between predicates on the state of programs. In this section, an equivalent relation, $\mathcal{L}_\mathcal{E}$, is defined, but instead of predicates, it is defined as a relation between subsets of states in u ($\mathcal{L}_\mathcal{E} \subseteq \mathbb{P}(u) \times \mathbb{P}(u)$). Any pair $a \mapsto b$ in $\mathcal{L}_\mathcal{E}$ indicates that the system reaches a state in b, when its execution arrives at any state in a. For this reason $\mathcal{L}_\mathcal{E}$ is named the *reachability* relation.

The definition of $\mathcal{L}_\mathcal{E}$ is given by induction. The base case needs the definition of the basic relation $\mathcal{E}$. At this time $\mathcal{E}$ cannot be defined. As indicated in section 3.1.1, basic liveness properties depends on fairness assumptions. $\mathcal{E}$ will be defined in the following sections according to minimal progress or weak fairness assumptions. However, these definitions must satisfy two requirements. The first requirement is as follows: If $a \subseteq b$, for any a and b in $\mathbb{P}(u)$, then $a \mapsto b \in \mathcal{E}$ must hold. The second requirement relates $\mathcal{E}$ with the set transformer W: For any ordered pair $a \mapsto b \in \mathcal{E}$, the inclusion $a \cap \overline{b} \subseteq W(b)$ must hold. This inclusion indicates that any execution of W starting in $a \cap \overline{b}$, terminates into a state of b.

Definition 4.4 (Reachability Relation)
The reachability *relation* $\mathcal{L}_\mathcal{E}$, $\mathcal{L}_\mathcal{E} \in \mathbb{P}(u) \leftrightarrow \mathbb{P}(u)$, *is defined by the following induction scheme:*
(**SBR**)*:* $\mathcal{E} \subseteq \mathcal{L}_\mathcal{E}$
(**STR**)*:* $\mathcal{L}_\mathcal{E}$ **;** $\mathcal{L}_\mathcal{E} \subseteq \mathcal{L}_\mathcal{E}$
(**SDR**)*:* $\forall (q, l) \cdot (q \in \mathbb{P}(u) \wedge l \subseteq \mathbb{P}(u) \Rightarrow (l \times \{q\} \subseteq \mathcal{L}_\mathcal{E} \Rightarrow \bigcup(l) \mapsto q \in \mathcal{L}_\mathcal{E}))$
Closure*:* $\forall l' \cdot (l' \in u \leftrightarrow u \wedge \mathcal{E} \subseteq l' \wedge l' \,;\, l' \subseteq l' \wedge$
$\forall (q, l) \cdot (q \in \mathbb{P}(u) \wedge l \subseteq \mathbb{P}(u) \wedge l \times \{q\} \subseteq l' \Rightarrow \bigcup(l) \mapsto q \in l') \Rightarrow \mathcal{L}_\mathcal{E} \subseteq l')$

$\bigcup(l)$ in the SDR rule and the closure clause, denotes the generalized union of subsets in l. Rules SBR, STR and SDR are the set counterparts of the basic rule for *leads-to* BRL, transitivity rule TRA and disjunction rule DSJ respectively, as defined in section 3.1.2.

The following equivalence connects the reachability relation $\mathcal{L}_\mathcal{E}$ with the *leads-to* relation of UNITY logic:

$$P(x) \rightsquigarrow Q(x) \equiv \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_\mathcal{E} \tag{4.7}$$

The proof of this equivalence is given in appendix A; the proof uses the fact that the property $P \rightsquigarrow Q$ in UNITY is equivalent to $P \wedge I \rightsquigarrow Q \wedge I$, considering I as an invariant of S, because the *leads-to* relation is defined in states reachable from the initial conditions [62].

Soundness and Completeness

We are now ready to state our main theorem, formally indicating that *termination* and *reachability* relations are equal:

Theorem 2 (Soundness and Completeness)
Let W be a monotonic and strict set transformer and $\mathcal{F}(r) = (\overline{r} \Longrightarrow W)$ for any r in $\mathbb{P}(u)$. Let relations $\mathcal{T}$ and $\mathcal{L}_{\mathcal{E}}$ be defined as definitions 4.3 and 4.4 respectively. Considering (a) $a \mapsto b \in \mathcal{E} \Rightarrow a \cap \overline{b} \subseteq W(b)$, (b) $a \subseteq b \Rightarrow a \mapsto b \in \mathcal{E}$ and (c) $W(r) \mapsto r \in \mathcal{L}_{\mathcal{E}}$, for any a, b and r in $\mathbb{P}(u)$, the following equality holds:

$$\mathcal{L}_{\mathcal{E}} = \mathcal{T}$$

Premise (a) and (b) were commented in the previous section. Premise (c) asserts that any set r is reached from the set $W(r)$ which is the largest subset of states where a step of the iteration terminates in r.

The proof of this theorem is given in two parts: first we prove the inclusion $\mathcal{L}_{\mathcal{E}} \subseteq \mathcal{T}$ and then $\mathcal{T} \subseteq \mathcal{L}_{\mathcal{E}}$.

Proof of $\mathcal{L}_{\mathcal{E}} \subseteq \mathcal{T}$ The proof of this inclusion follows from the closure clause in definition 4.4, particularizing the quantified variable l' to relation $\mathcal{T}$. Then $\mathcal{L}_{\mathcal{E}} \subseteq \mathcal{T}$ follows from $\mathcal{E} \subseteq \mathcal{T}$, $\mathcal{T};\mathcal{T} \subseteq \mathcal{T}$ and $l \times \{q\} \subseteq \mathcal{T} \Rightarrow \bigcup(l) \mapsto q \in \mathcal{T}$ for any l in $\mathbb{P}(\mathbb{P}(u))$ and q in $\mathbb{P}(u)$.

The proof of $\mathcal{E} \subseteq \mathcal{T}$ uses the following property for monotonic function f and iteration defined in (4.5):

$$\forall \alpha \cdot (f^{\alpha} \subseteq \mathsf{fix}(f)) \tag{4.8}$$

which is easily proved by transfinite induction; the proof is given in appendix (C.1). The proof of $\mathcal{E} \subseteq \mathcal{T}$ is given by the proof of $a \mapsto b \in \mathcal{E} \Rightarrow a \mapsto b \in \mathcal{T}$:

1.	$a \mapsto b \in \mathcal{E}$	; premise
2.	$a \cap \overline{b} \subseteq W(b)$	; 1 and hyp. (a)
3.	$a \subseteq \mathcal{F}(b)(b)$	; 2 and def. (4.4)
4.	$a \subseteq \mathcal{F}(b)^2$	; 3 and iterate (4.5)
5.	$a \subseteq \mathsf{fix}(\mathcal{F}(b))$	; 4 and (4.8)
6.	$a \mapsto b \in \mathcal{T}$	; 5 and def. (4.6)

The following property is needed in the proof of transitivity of $\mathcal{T}$:

$$a \mapsto b \in \mathcal{T} \Rightarrow \mathsf{fix}(\mathcal{F}(a)) \subseteq \mathsf{fix}(\mathcal{F}(b)) \tag{4.9}$$

for any a and b in $\mathbb{P}(u)$. Taking $a \mapsto b \in \mathcal{T}$ as a premise, and considering $\mathsf{fix}(\mathcal{F}(a))$ as the least fixpoint of $\mathcal{F}(a)$, in order to prove property (4.9) it suffices to prove

$$\mathcal{F}(a)(\mathsf{fix}(\mathcal{F}(b))) \subseteq \mathsf{fix}(\mathcal{F}(b))$$

which follows directly from $a \mapsto b \in \mathcal{T}$ and $\mathcal{F}(b)(\mathsf{fix}(\mathcal{F}(b))) = \mathsf{fix}(\mathcal{F}(b))$. Now the proof of $\mathcal{T}\,;\mathcal{T} \subseteq \mathcal{T}$ is equivalent to prove $a \mapsto b \in \mathcal{T}\,;\mathcal{T} \Rightarrow a \mapsto b \in \mathcal{T}$ for any a and b in $\mathbb{P}(u)$:

1.	$\exists c \cdot (a \mapsto c \in \mathcal{T} \wedge c \mapsto b \in \mathcal{T})$	; from $a \mapsto b \in \mathcal{T};\mathcal{T}$
2.	$\exists c \cdot (a \subseteq \mathsf{fix}(\mathcal{F}(c)) \wedge c \mapsto b \in \mathcal{T})$	; 1 and def. $\mathcal{T}$
3.	$\exists c \cdot (a \subseteq \mathsf{fix}(\mathcal{F}(c)) \wedge \mathsf{fix}(\mathcal{F}(c)) \subseteq \mathsf{fix}(\mathcal{F}(b)))$	; 2 and (4.9)
4.	$a \subseteq \mathsf{fix}(\mathcal{F}(b))$	; 3
5.	$a \mapsto b \in \mathcal{T}$	; 6 and def. $\mathcal{T}$

Finally, the proof of $l \times \{q\} \subseteq \mathcal{T} \Rightarrow \bigcup(l) \mapsto q \in \mathcal{T}$ is as follows:

1.	$l \times \{q\} \subseteq \mathcal{T}$	; premise
2.	$\forall p \cdot (p \in l \Rightarrow p \mapsto q \in \mathcal{T})$	; 1
3.	$\forall p \cdot (p \in l \Rightarrow p \subseteq \mathsf{fix}(\mathcal{F}(q)))$	; 2 and def. $\mathcal{T}$
4.	$\bigcup(l) \subseteq \mathsf{fix}(\mathcal{F}(q))$	; 3
5.	$\bigcup(l) \mapsto q \in \mathcal{T}$	; 4 and def. $\mathcal{T}$

This last deduction concludes the proof of $\mathcal{L}_{\mathcal{E}} \subseteq \mathcal{T}$.

Proof of $\mathcal{T} \subseteq \mathcal{L}_{\mathcal{E}}$ The proof of this inclusion requires the following property:

$$\forall r \cdot (r \in \mathbb{P}(u) \Rightarrow \mathcal{F}(r)^{\alpha} \mapsto r \in \mathcal{L}_{\mathcal{E}}) \quad \text{for any ordinal } \alpha \tag{4.10}$$

The proof of (4.10) is done by transfinite induction. For a successor ordinal, the following implication must be proved:

$$\mathcal{F}(r)^{\alpha} \mapsto r \in \mathcal{L}_{\mathcal{E}} \Rightarrow \mathcal{F}(r)^{\alpha+1} \mapsto r \in \mathcal{L}_{\mathcal{E}}$$

the proof for this case is given in appendix C.1. For a limit ordinal, the following implication must be proved:

$$\forall \beta \cdot (\beta < \alpha \Rightarrow \mathcal{F}(r)^{\beta} \mapsto r \in \mathcal{L}_{\mathcal{E}}) \Rightarrow \mathcal{F}(r)^{\alpha} \mapsto r \in \mathcal{L}_{\mathcal{E}}$$

The proof is as given next:

1.	$\forall \beta \cdot (\beta < \alpha \Rightarrow \mathcal{F}(r)^{\beta} \mapsto r \in \mathcal{L}_{\mathcal{E}})$	; ind. hyp.
2.	$\forall \beta \cdot (\beta < \alpha \Rightarrow W(\mathcal{F}(r)^{\beta}) \mapsto \mathcal{F}(r)^{\beta} \in \mathcal{L}_{\mathcal{E}})$	; from hyp. (c)
3.	$\forall \beta \cdot (\beta < \alpha \Rightarrow W(\mathcal{F}(r)^{\beta}) \mapsto r \in \mathcal{L}_{\mathcal{E}})$	; 2, 1 and STR
4.	$r \mapsto r \in \mathcal{L}_{\mathcal{E}}$	; hyp. (b) and SBR
5.	$\forall \beta \cdot (\beta < \alpha \Rightarrow r \cup W(\mathcal{F}(r)^{\beta}) \mapsto r \in \mathcal{L}_{\mathcal{E}})$	; 4, 3 and SDR
6.	$\forall \beta \cdot (\beta < \alpha \Rightarrow \mathcal{F}(r)(\mathcal{F}(r)^{\beta}) \mapsto r \in \mathcal{L}_{\mathcal{E}})$	; def. $\mathcal{F}(r)$ and 5
7.	$\bigcup \beta \cdot (\beta < \alpha \mid \mathcal{F}(r)(\mathcal{F}(r)^{\beta})) \mapsto r \in \mathcal{L}_{\mathcal{E}}$	; 6 and SDR
8.	$\mathcal{F}(r)^{\alpha} \mapsto r \in \mathcal{L}_{\mathcal{E}}$	; 7 and def. iterate

Using (4.10), the inclusion $\mathcal{T} \subseteq \mathcal{L}_{\mathcal{E}}$ is demonstrated by the proof of $a \mapsto b \in \mathcal{T} \Rightarrow a \mapsto b \in \mathcal{L}_{\mathcal{E}}$ for any a and b in $\mathbb{P}(u)$ as follows:

1.	$a \subseteq \mathsf{fix}(\mathcal{F}(b))$	; from $a \mapsto b \in \mathcal{T}$
2.	$\exists \alpha \cdot (a \subseteq \mathcal{F}(b)^{\alpha})$	; 1 and theorem 1
3.	$\exists \alpha \cdot (a \mapsto \mathcal{F}(b)^{\alpha} \in \mathcal{L}_{\mathcal{E}})$	; 2, (b) and SBR
4.	$\exists \alpha \cdot (a \mapsto \mathcal{F}(b)^{\alpha} \in \mathcal{L}_{\mathcal{E}} \wedge \mathcal{F}(b)^{\alpha} \mapsto b \in \mathcal{L}_{\mathcal{E}})$	; 3 and (4.10)
5.	$a \mapsto b \in \mathcal{L}_{\mathcal{E}}$	; 4 and STR

This deduction concludes the proof of theorem 2.

4.2.2 Link with Traces Semantics

At this point, the meaning of *leads-to* properties is given in terms of the least fixpoint of the set transformer $\mathcal{F}(b)$ for certain subset b of u. This set transformer denotes a step in the computations of the system if it is executed in a state where $\overline{b}$ holds. Therefore, $\mathsf{fix}(\mathcal{F}(b))$ denotes the set of states where any computation can start and terminates in a state where b holds.

For any set transformer F, the associated before-after relation can be defined as in [7], for any subset p of u, by:

$$\mathsf{rel}(F)^{-1}[p] = \overline{F(\overline{p})}$$

Any pair $\sigma \mapsto \tau$ in this relation, indicates that execution of F in a state σ can reach the state τ. In the special case where $p = u$ in the above definition, we note that the guard of F corresponds to the domain of $\mathsf{rel}(F)$, that is, to the set of states where F is enabled. Therefore $\mathsf{rel}(F)^*$, the transitive and reflexive closure of $\mathsf{rel}(F)$, contains all pairs $\sigma \mapsto \tau$, where τ can be reached from a repeated execution of F starting in σ. Moreover, from [2], the before-after relation of the iteration operator is given by:

$$\mathsf{rel}(F^{\frown}) = \overline{\mathsf{pre}(F^{\frown})} \times u \cup \mathsf{rel}(F)^*$$

, where $\mathsf{pre}(F^{\frown})$, the set where termination of the iteration of F is guaranteed, is equal to $\mathsf{fix}(F)$. From this equality, we can observe that the before-after relation of $F^{\frown}$ contains the transitive and reflexive closure of $\mathsf{rel}(F)$, as well as the Cartesian product $\{\delta\} \times u$, for any set δ where the iteration does not terminate. If for any different states σ and τ, such that $\sigma \mapsto \tau \in \mathsf{rel}(F)^*$ holds, we can conclude that there is a sequence of states, or state traces, $\sigma_0, \sigma_1, \ldots, \sigma_n$ where $\sigma_0 = \sigma$, $\sigma_n = \tau$ and for any $i \geq 0$, $\sigma_i \mapsto \sigma_{i+1} \in \mathsf{rel}(F)$ holds. Any state sequence starting in a state in $\overline{\mathsf{pre}(F^{\frown})}$, is a divergent sequence. On the other hand, if the sequence starts in a state in $\mathsf{pre}(F^{\frown})$, the sequence is finite and its last state is in $\overline{\mathsf{dom}(\mathsf{rel}(F))}$.

From the above discussion, in a way similar to [9], a semantic function in terms of state traces can be associated to $\mathcal{F}(b)$, for any σ in u, as follows:

$$\mathcal{M}[\![\mathcal{F}(b)]\!](\sigma) = \{\tau \mid \tau \in b \wedge \sigma \mapsto \tau \in \mathsf{rel}(\mathcal{F}(b))^*\} \cup \{\bot \mid \sigma \in \overline{\mathsf{fix}(\mathcal{F}(b))}\}$$

where the symbol $\bot$ is a special state denoting divergence. Now, if for any subset a of u, the inclusion $\mathcal{M}[\![\mathcal{F}(b)]\!](a) \subseteq b$ holds [1], that means that the iteration of events in the system leads to a state into b when its execution starts in a state in a. This interpretation of the inclusion allows us to define the truth value of a *leads-to* property: for any predicate P and Q, a liveness property $P \rightsquigarrow Q$ holds in a system ($\models P \rightsquigarrow Q$) if $\mathcal{M}[\![\mathcal{F}(b)]\!](\mathsf{set}(P)) \subseteq \mathsf{set}(Q)$ holds.

Considering the set of pairs $a \mapsto b$ satisfying $\mathcal{M}[\![\mathcal{F}(b)]\!](a) \subseteq b$, allow us to obtain an equivalent definition of the termination relation $\mathcal{T}$ presented in section 4.2.1. In this way, soundness of the rules BRL, TRA and DSJ, means that if $P \rightsquigarrow Q$ is derived from these rules ($\vdash \mathsf{set}(P) \mapsto \mathsf{set}(Q) \in \mathcal{L}_{\mathcal{E}}$), then $\models P \rightsquigarrow Q$ holds. Relative completeness of these rules is proved by the implication $\models P \rightsquigarrow Q \Rightarrow \vdash \mathsf{set}(P) \mapsto \mathsf{set}(Q) \in \mathcal{L}_{\mathcal{E}}$. These results are stated and proved in theorem 2 by the equality $\mathcal{L}_{\mathcal{E}} = \mathcal{T}$.

Finally we note that the instantiation of $\mathcal{F}(b)$ to $\mathcal{F}_m(b)$ or $\mathcal{F}_w(b)$, as it will be done in the following sections, allows a rigorous definition of the semantic function under minimal progress or weak fairness assumptions respectively.

[1] For any subset t of u, $\mathcal{M}[\![\mathcal{F}(b)]\!](t) = \bigcup_{\sigma \in t} \mathcal{M}[\![\mathcal{F}(b)]\!](\sigma)$.

4.2.3 Minimal Progress

In this paragraph the *termination* and *reachability* relations under minimal progress are defined and it is proved that they satisfy the premises of theorem 2. Therefore we prove that relations $\mathcal{T}$ and $\mathcal{L}_{\mathcal{E}}$ are equal in the case of minimal progress.

Termination under MP

To model a step of the iteration of events of system S under minimal progress assumptions, we note that if we need to establish a certain postcondition when this step is achieved, any event in S must be able to establish the postcondition. Moreover, as we are interested in the execution of any event, we need to start the execution step in a state satisfying the guard of at least one event. Therefore, taking into account these considerations, we propose the following preconditioned set transformer:

$$W_m = \mathsf{grd}(S) \mid S \tag{4.11}$$

From the definition of the preconditioned set transformer in section 4.1.1 follows $W_m(r) = \mathsf{grd}(S) \cap S(r)$. From this derivation, the strictness of W_m is proved: $W_m(\varnothing) = (\mathsf{grd}(S) \cap S(\varnothing)) = \varnothing$. On another hand, the monotonicity of W_m is proved from the monotonicity of S.

The body of the iteration of events under minimal progress is the guarded event $\mathcal{F}_m(r)$ defined as follows:

$$\mathcal{F}_m(r) = \overline{r} \Longrightarrow W_m \tag{4.12}$$

Definition of the *termination* relation under minimal progress is given by all ordered pairs $a \mapsto b$ satisfying $a \subseteq \mathsf{pre}(\mathcal{F}_m(b)^{\frown})$:

$$\mathcal{T}_m = \{\, a \mapsto b \,|\, a \subseteq u \wedge b \subseteq u \wedge a \subseteq \mathsf{fix}(\mathcal{F}_m(b)) \,\} \tag{4.13}$$

Reachability under MP

Recalling that x is the state variable transformed by S, the basic relation under minimal progress is defined as the relation containing all ordered pairs $a \mapsto b$ from which the property $x \in a \gg_m x \in b$ can be derived:

$$\mathcal{E}_m = \{\, a \mapsto b \,|\, a \subseteq u \wedge b \subseteq u \wedge a \cap \overline{b} \subseteq S(b) \cap \mathsf{grd}(S) \,\} \tag{4.14}$$

From definitions of $\mathcal{E}_m$ and W_m follows the proof of premise (a) of theorem 2 for the case of minimal progress $a \mapsto b \in \mathcal{E}_m \Rightarrow a \cap \overline{b} \subseteq W_m(b)$:

1. $a \mapsto b \in \mathcal{E}_m$; premise
2. $a \cap \overline{b} \subseteq \mathsf{grd}(S) \cap S(b)$; 1 and def. (4.14)
3. $a \cap \overline{b} \subseteq (\mathsf{grd}(S) \mid S)(b)$; 2 and set transf.
4. $a \cap \overline{b} \subseteq W(b)$; 3 and def. (4.11)

From definition of $\mathcal{E}_m$, the implication $a \subseteq b \Rightarrow a \mapsto b \in \mathcal{E}_m$ follows immediately because $a \cap \overline{b} = \varnothing$. It proves premise (b) of theorem 2 for the case of minimal progress.

Now, we use an induction scheme to define the *reachability* relation under minimal progress $\mathcal{L}_{\mathcal{E}_m}$ similar to definition 4.4. Therefore $\mathcal{L}_{\mathcal{E}_m}$ is the smallest relation containing the base relation $\mathcal{E}_m$ and it is both, transitive and disjunctive.

Finally we prove that the weakest precondition $W_m(r)$, for any $r \in \mathbb{P}(u)$ leads to r: $W_m(r) \mapsto r \in \mathcal{L}_{\mathcal{E}_m}$

1. $\mathsf{grd}(S) \cap S(r) \cap \overline{r} \subseteq \mathsf{grd}(S) \cap S(r)$; trivial
2. $\mathsf{grd}(S) \cap S(r) \mapsto r \in \mathcal{E}_m$; 1 and def. $\mathcal{E}_m$
3. $W_m(r) \mapsto r \in \mathcal{E}_m$; 2 and (4.11)
4. $W_m(r) \mapsto r \in \mathcal{L}_{\mathcal{E}_m}$; 3 and def. $\mathcal{L}_{\mathcal{E}_m}$

It proves premise (c) of theorem 2 for the case of minimal progress.

At this time, monotonicity and strictness of W_m and premises (a), (b) and (c) of theorem 2 instantiated to the case of minimal progress have been proved. Therefore the equality between *termination* and *reachability* relations is stated:

$$\mathcal{T}_m = \mathcal{L}_{\mathcal{E}_m} \tag{4.15}$$

4.2.4 Weak Fairness

In this subsection, the *termination* and *reachability* relations are defined for weak fairness assumptions. We prove that premises of theorem 2, instantiated to the case of weak fairness assumptions, are satisfied with these definitions. Therefore the equality between these relations is proved.

Termination under WF

The dovetail operator, presented in section 4.1.1, is used to model a *fair loop* for a certain event G in E:

$$Y(q)(G) = \overline{q} \Longrightarrow ((S \,;\, Y(q)(G)) \,\triangledown\, (\mathsf{grd}(G) \mid G)) \tag{4.16}$$

The guard $\overline{q}$ of this loop prevents iteration of the fair choice in any state belonging to q. Informally, we expect that any execution of $Y(G)(q)$ in any state in $q \cup \mathsf{grd}(G)$ terminates. Execution of $Y(q)(G)$ in $\overline{q} \cap \mathsf{grd}(G)$ cannot loop forever because the dovetail operator prevents unlimited execution of the branch $S \,;\, Y(q)(G)$. Moreover, the set transformer $\mathsf{grd}(G) \mid G$ is always enabled ($\mathsf{grd}(\mathsf{grd}(G) \mid G) = u$) and therefore it will be eventually executed. All our claims are formally justified by the calculi of termination set and the liberal weakest precondition of $Y(q)(G)$, for any q and r, $r \neq u$ in $\mathbb{P}(u)$:

$$\mathsf{pre}(Y(q)(G)) = \mathsf{fix}(\overline{q} \cap G(\varnothing) \Longrightarrow \overline{S(q)} \mid S) \tag{4.17}$$

$$\mathcal{L}(Y(q)(G))(r) = \mathsf{FIX}(\overline{q} \Longrightarrow (\mathsf{grd}(G) \cap G(r) \mid S)) \tag{4.18}$$

These calculi follow from definitions of set transformers given in section 4.1.1 and the extreme solutions of the recursive equations generated. The proof of (4.17) and (4.18) are given in appendices C.2.1 and C.2.2 respectively. Moreover, in appendix C.2.3 appears the proof of the following inclusion, for any q and r in $\mathbb{P}(u)$:

$$\mathcal{L}(Y(q)(G))(r) \subseteq \mathsf{pre}(Y(q)(G)) \tag{4.19}$$

(4.19), and the pairing conditions, give us the set transformer associated with the fair loop:

$$Y(q)(G)(r) = \mathsf{FIX}(\overline{q} \Longrightarrow (\mathsf{grd}(G) \cap G(r) \mid S)) \tag{4.20}$$

From this definition follows the monotonicity of $Y(q)(G)$, which is proved in appendix C.2.

The fair loop $Y(q)(G)$ models a *fair G-step* in the iteration of events under weak fairness assumptions. We say that G is the helpful event in this G-step. A *fair step* in the iteration of events is modeled by the following set transformer:

$$W_w = \lambda r \cdot (r \subseteq u \mid \bigcup G \cdot (G \in E \mid Y(r)(G)(r))) \tag{4.21}$$

From (4.20) follows $\mathsf{grd}(Y(q)(G)) = \overline{q}$ for any G in E and $q \in \mathbb{P}(u)$, therefore the strictness of W_w follows. On the other hand, from monotonicity of $Y(q)(G)$ follows the monotonicity of W_w. These three proofs are given in appendix C.2.

The body of the iteration of events under weak fairness is the guarded event $\mathcal{F}_w(r)$ defined as follows:

$$\mathcal{F}_w(r) = \overline{r} \Longrightarrow W_w \tag{4.22}$$

Definition of the *termination* relation under weak fairness is:

$$\mathcal{T}_w = \{\, a \mapsto b \,|\, a \subseteq u \wedge b \subseteq u \wedge a \subseteq \mathsf{fix}(\mathcal{F}_w(b)) \,\} \tag{4.23}$$

Reachability under WF

The basic relation $\mathcal{E}(G)$, for a helpful event G, is defined as the set of pairs $a \mapsto b$ from which a property $G \cdot x \in a \gg_w x \in b$ can be defined:

$$\mathcal{E}(G) = \{\, a \mapsto b \,|\, a \subseteq u \wedge b \subseteq u \wedge a \cap \overline{b} \subseteq S(a \cup b) \cap \overline{G(\varnothing)} \cap G(b) \,\} \tag{4.24}$$

Now, the basic relation for weak fairness is:

$$\mathcal{E}_w = \bigcup G \cdot (G \in E \mid \mathcal{E}(G)) \tag{4.25}$$

The proof of premise (a) of theorem 2 instantiated to weak fairness requires the following property which is proved in appendix C.2.8:

$$\forall G \cdot (G \in E \wedge a \mapsto b \in \mathcal{E}(G) \Rightarrow a \subseteq Y(b)(G)(b)) \tag{4.26}$$

Using (4.26), the proof of $a \mapsto b \in \mathcal{E}_w \Rightarrow a \cap \overline{b} \subseteq W_w(b)$ is:

1. $\forall G \cdot (G \in E \Rightarrow (a \mapsto b \in \mathcal{E}(G) \Rightarrow a \subseteq Y(b)(G)(b)))$; (4.26)
2. $\exists G \cdot (G \in E \wedge a \mapsto b \in \mathcal{E}(G)) \Rightarrow a \subseteq W_w(b)$; 1 and (4.21).
3. $a \mapsto b \in \bigcup G \cdot (G \in E \mid \mathcal{E}(G)) \Rightarrow a \subseteq W_w(b)$; 2
4. $a \mapsto b \in \mathcal{E}_w \Rightarrow a \subseteq W_w(b)$; 3 and (4.25)
5. $a \mapsto b \in \mathcal{E}_w \Rightarrow a \cap \overline{b} \subseteq W_w(b)$; 4 and $b \subseteq W_w(b)$

From (4.24) immediately follows $a \mapsto b \in \mathcal{E}(G)$, for any G in E if $a \subseteq b$ holds, and from (4.25) follows $a \subseteq b \Rightarrow a \mapsto b \in \mathcal{E}_w$. It proves premise (b) of theorem 2.

We use an induction scheme to define the *reachability* relation under weak fairness $\mathcal{L}_{\mathcal{E}_w}$ similar to definition 4.4. Therefore $\mathcal{L}_{\mathcal{E}_w}$ is the smallest relation containing the base relation $\mathcal{E}_w$ and it is both, transitive and disjunctive.

From (4.20) and (4.24) follows the following property proved in appendix C.2.9:

$$\forall(G, r) \cdot (G \in E \wedge r \subseteq u \Rightarrow Y(r)(G)(r) \mapsto r \in \mathcal{E}(G)) \tag{4.27}$$

This property is used to prove the premise (c) of theorem 2: $W_w(r) \mapsto r \in \mathcal{L}_{\mathcal{E}_w}$ as follows:

1. $\forall G \cdot (G \in E \Rightarrow Y(r)(G)(r) \mapsto r \in \mathcal{E}(G))$; from (4.27)
2. $\forall G \cdot (G \in E \Rightarrow \mathcal{E}(G) \subseteq \mathcal{E}_w)$; def. $\mathcal{E}_w$
3. $\forall G \cdot (G \in E \Rightarrow Y(r)(G)(r) \mapsto r \in \mathcal{E}_w)$; 2 and 1
4. $\forall G \cdot (G \in E \Rightarrow Y(r)(G)(r) \mapsto r \in \mathcal{L}_{\mathcal{E}_w})$; def. $\mathcal{L}_{\mathcal{E}_w}$
5. $\{\, Y(r)(G)(r) \mid G \in E \,\} \times \{r\} \subseteq \mathcal{L}_{\mathcal{E}_w}$; 4
6. $\bigcup(\{\, Y(r)(G)(r) \mid G \in E \,\}) \mapsto r \in \mathcal{L}_{\mathcal{E}_w}$; 5 and SDR
7. $\bigcup G \cdot (G \in E \mid Y(r)(G)(r)) \mapsto r \in \mathcal{L}_{\mathcal{E}_w}$; 6
8. $W_w(r) \mapsto r \in \mathcal{L}_{\mathcal{E}_w}$; 7 and def. F

At this time *termination* ($\mathcal{T}_w$), basic relation ($\mathcal{E}_w$) and *reachability* ($\mathcal{L}_{\mathcal{E}_m}$) relations for weak fairness assumptions have been defined. Monotonicity and strictness of the set transformer W_w, and premises (a), (b) and (c) of theorem 2 instantiated to the case of weak fairness have been proved. Therefore, the equality between *termination* and *reachability* relations under weak fairness is stated:

$$\mathcal{T}_w = \mathcal{L}_{\mathcal{E}_w} \tag{4.28}$$

4.3 Soundness of Rules for Basic Properties

In this section we prove soundness of the rules for basic liveness properties under minimal progress and weak fairness assumptions. Soundness of these rules are proved if the sufficient conditions for a basic liveness property $P \gg Q$ guarantee that iteration of events, under a certain fairness assumption, in a state satisfying the invariant and P, terminates into a state of Q.

Soundness for Minimal Progress

In fact, soundness of proof obligations for basic properties under minimal progress has been already proved. In the previous section, the inclusion $\mathcal{L}_{\mathcal{E}_m} \subseteq \mathcal{T}_m$ is proved. This inclusion demonstrates soundness of the rules for the *leads-to* relation. Now, by the inductive definition of $\mathcal{L}_{\mathcal{E}_m}$ follows that the reachability relation contains the basic relation: $\mathcal{E}_m \subseteq \mathcal{L}_{\mathcal{E}_m}$. Then by transitivity of set inclusion follows that the basic relation is contained in the termination relation: $\mathcal{E}_m \subseteq \mathcal{T}_m$. In the following paragraphs we show how this inclusion is related to the proof of soundness of MP0 and MP1.

From definitions of the basic (4.14) and termination (4.13) relations under minimal progress assumption and the inclusion $\mathcal{E}_m \subseteq \mathcal{T}_m$ follows:

$$\begin{aligned}&\forall(p, q) \cdot (p \in \mathbb{P}(u) \wedge q \in \mathbb{P}(u) \wedge p \cap \overline{q} \subseteq S(q) \cap \mathsf{grd}(S) \Rightarrow \\ &\quad p \subseteq \mathsf{pre}((\overline{q} \Longrightarrow \mathsf{grd}(S) \mid S)^\frown))\end{aligned} \tag{4.29}$$

This implication states that iteration of S in any state satisfying p terminates into a state of q, for any subset p and q of u, satisfying the premises of (4.29). Moreover, from $p \cap \overline{q} \subseteq S(q)$ in the premise, follows that only one step is needed to terminate the iteration of events under a minimal progress assumption, which is the intended meaning of a basic property $x \in p \gg_m x \in q$ under a weak fairness assumption.

In order to demonstrate soundness of the rules MP0 and MP1, proposed in section 3.2, it suffices to note that the sets p and q satisfying the premise of (4.29) are the sets satisfying both predicates $I \wedge P$ and $I \wedge Q$ respectively, where P and Q are the predicates satisfying MP0 and MP1 rules. The formal proof of this fact requires to define:

$$p = \{\, z \mid I(z) \wedge P(z) \,\} \quad \text{and} \quad q = \{\, z \mid I(z) \wedge Q(z) \,\}$$

Now, we prove that the conjunction of $MP0$ and $MP1$ is equivalent to the premise of (4.29):

$$\begin{array}{ll}
& \text{MP0} \wedge \text{MP1} \\
\equiv & \\
& \forall x \cdot (I(x) \wedge P(x) \wedge \neg Q(x) \Rightarrow ([S]\, Q(x)) \wedge \mathit{grd}(S)) \\
\equiv & \qquad \{\ I \Rightarrow [S]\ I \text{, conjunctive } S \text{ and def. } \mathit{grd}\ \} \\
& \forall x \cdot (I(x) \wedge P(x) \wedge \neg Q(x) \Rightarrow ([S]\, Q(x) \wedge I(x)) \wedge \neg [S]\, \mathit{false}) \\
\equiv & \qquad \{\ \text{def. of } p \text{ and } q\ \} \\
& \forall x \cdot (x \in p \cap \overline{q} \Rightarrow ([S]\, x \in q) \wedge \neg [S]\, x \in \varnothing) \\
\equiv & \qquad \{\ \text{def. of set transformer}\ \} \\
& \forall x \cdot (x \in p \cap \overline{q} \Rightarrow x \in S(q) \cap \overline{S(\varnothing)}) \\
\equiv & \\
& p \cap \overline{q} \subseteq S(q) \cap \mathsf{grd}(S)
\end{array}$$

Soundness for Weak Fairness

As for minimal progress, soundness of the rules for basic liveness properties under weak fairness has been already proved. However, the proof is not as explicit as the case of minimal progress. The following paragraphs gives more insight into that proof.

The previous section proves that the basic relation under weak fairness assumptions is contained in the termination relation: $\mathcal{E}_w \subseteq \mathcal{T}_w$. On another hand, for a helpful event G of E, the basic relation $\mathcal{E}(G)$ is contained in $\mathcal{E}_w$; therefore $\mathcal{E}(G)$ is contained in the termination relation $\mathcal{T}_w$: $\mathcal{E}(G) \subseteq \mathcal{T}_w$. From this last inclusion and the definition of $\mathcal{T}_w$ (4.23) follows:

$$\forall (p, q) \cdot (p \mapsto q \in \mathcal{E}(G) \Rightarrow p \subseteq \mathsf{fix}(\mathcal{F}_w(q)))$$

This implication states that if the pair $p \mapsto q$ is included in the basic relation $\mathcal{E}(G)$, then any state of p guarantees that iteration of fair loops $Y(q)(G')$, for any event G' of E, terminates into a state in q. This interpretation does not give information about the number of iterations for a particular loop $Y(q)(G)$. This is the intended meaning for the *leads-to* relation. However, the intended meaning of *ensures* under a weak fairness assumption, requires only one iteration of the fair loop $Y(q)(G)$. In fact, this has been already proved with the implication (4.26) in page 81:

$$\forall G \cdot (G \in E \wedge p \mapsto q \in \mathcal{E}(G) \Rightarrow p \subseteq Y(q)(G)(q)) \tag{4.26}$$

This implication states that for any pair $p \mapsto q$ in the basic relation $\mathcal{E}(G)$, the fair loop, starting its execution in any state of p, terminates into a state of q by the execution of G.

In order to prove soundness of the rules WF0 and WF1 for basic properties under a weak fairness assumption, it suffices to note that the sets p and q, satisfying the premise $p \mapsto q \in \mathcal{E}(G)$ of (4.26), are the sets containing the states satisfying the invariant, and the predicates P and Q respectively of WF0 and WF1. The formal proof requires the definition of p and q as before:

$$p = \{\, z \mid I(z) \wedge P(z) \,\} \quad \text{and} \quad q = \{\, z \mid I(z) \wedge Q(z) \,\}$$

and the definition of the basic relation $\mathcal{E}(G)$ (4.24):

$$\begin{array}{ll}
& \text{WF0} \wedge \text{WF1} \\
\equiv & \\
& \forall x \cdot (I(x) \wedge P(x) \wedge \neg Q(x) \Rightarrow ([S]\,(P \vee Q)) \wedge \mathit{grd}(G) \wedge [G]\,Q) \\
\equiv & \qquad \{\ I \Rightarrow [S]\,I, \text{ def. } \mathit{grd} \text{ and conjunctive } S \text{ and } G\ \} \\
& \forall x \cdot (I(x) \wedge P(x) \wedge \neg Q(x) \Rightarrow ([S]\,((P \vee Q) \wedge I)) \wedge \neg[G]\,\mathit{false} \wedge [G]\,(Q \wedge I)) \\
\equiv & \qquad \{\ \text{def. of } p \text{ and } q\ \} \\
& \forall x \cdot (x \in p \cap \overline{q} \Rightarrow ([S]\,x \in p \cup q) \wedge \neg[G]\,x \in \varnothing \wedge [G]\,x \in q) \\
\equiv & \qquad \{\ \text{def. of set transformer}\ \} \\
& \forall x \cdot (x \in p \cap \overline{q} \Rightarrow x \in S(p \cup q) \cap \overline{G(\varnothing)} \cap G(q)) \\
\equiv & \\
& p \cap \overline{q} \subseteq S(p \cup q) \cap \mathsf{grd}(G) \cap G(q)
\end{array}$$

4.4 Deriving Liveness Properties

In this section we present two examples where we show practical usefulness of equalities between *termination* and *reachability* relations under minimal progress and weak fairness assumptions. This section is divided in three parts. In the first part, we state and prove the *Variant Theorem*, which allows us to prove termination of iterations over a set transformer if a variant decreases. In the second part, we use this theorem to prove a sufficient condition allowing derivation of liveness properties under minimal progress. Finally, we give another sufficient condition to derive a liveness property under minimal progress when a similar property holds in weak fairness assumptions

4.4.1 The Variant Theorem

The variant theorem allows us to prove termination of iteration of conjunctive set transformers. This theorem considers a total function which maps each element of the state space to an element of a well founded order and a set which is invariant at each iteration of the set transformer. The theorem states that if any execution of the set transformer starting in a state in the invariant set and a certain value of the variant function, terminates in a state where the value of the variant is decremented, then the invariant set is contained in the termination set of the iteration of the set transformer. Formally, the theorem is stated as follows:

Theorem 3 (Variant Theorem)
Let $V \in u \to \mathbb{N}$, $v = \lambda n \cdot (n \in \mathbb{N} \mid \{\, z \mid z \in u \wedge V(z) = n \,\})$ *and* $v' = \lambda n \cdot (n \in \mathbb{N} \mid \{\, z \mid z \in u \wedge$

$V(z) < n\,\}$). *For any conjunctive set transformer* f *in* $\mathbb{P}(u) \to \mathbb{P}(u)$ *and* p *in* $\mathbb{P}(u)$, *such that* $v(n) \cap p \subseteq f(v'(n))$ *and* $p \subseteq f(p)$, *for any* n *in* $\mathbb{N}$, *the following inclusion holds:*

$$p \subseteq \mathit{fix}(f)$$

The proof of this theorem uses the following equalities:

$$\forall n \cdot (n \in \mathbb{N} \Rightarrow v'(n) = \bigcup i \cdot (i \in \mathbb{N} \wedge i < n \mid v(i))) \tag{4.30}$$

$$\bigcup i \cdot (i \in \mathbb{N} \mid v'(i+1)) = \bigcup i \cdot (i \in \mathbb{N} \mid v(i)) \tag{4.31}$$

$$\bigcup i \cdot (i \in \mathbb{N} \mid v(i)) = u \tag{4.32}$$

and the following property:

$$\forall n \cdot (n \in \mathbb{N} \Rightarrow \bigcup i \cdot (i \in \mathbb{N} \wedge i \leq n \mid v(i)) \cap p \subseteq f^{n+1}) \tag{4.33}$$

which are proved, under the assumptions of the theorem 3, in appendix C.3. The proof of theorem 3 is as follows:

1. $\forall n \cdot (n \in \mathbb{N} \Rightarrow \bigcup i \cdot (i \in \mathbb{N} \wedge i \leq n \mid v(i)) \cap p \subseteq \mathsf{fix}(f))$; (4.33) and (4.8)
2. $\forall n \cdot (n \in \mathbb{N} \Rightarrow v'(n+1) \cap p \subseteq \mathsf{fix}(f))$; from (4.30) and 1
3. $\bigcup i \cdot (i \in \mathbb{N} \mid v'(i+1)) \cap p \subseteq \mathsf{fix}(f)$; 2
4. $\bigcup i \cdot (i \in \mathbb{N} \mid v(i)) \cap p \subseteq \mathsf{fix}(f)$; 3 and (4.31)
5. $u \cap p \subseteq \mathsf{fix}(f)$; 4 and (4.32)
6. $p \subseteq \mathsf{fix}(p)$; 5 and $p \subseteq u$

4.4.2 A Sufficient Condition for Minimal Progress

A system reaches a certain set from any set of starting states under minimal progress, if the set of depart is invariant in the system, it is contained in the guard of the system and each execution of the system decrements a variant. Formally, these conditions are stated as follows:

ANTECEDENT	CONSEQUENT
$\forall n \cdot (n \in \mathbb{N} \Rightarrow a \cap \overline{b} \cap v(n) \subseteq S(v'(n)))$ $a \cap \overline{b} \subseteq \mathsf{grd}(S) \cap S(a)$	$a \mapsto b \in \mathcal{L}_{\mathcal{E}_m}$

We remark from definition (4.12), that $\mathcal{F}_m(b)$ is a conjunctive set transformer. From this remark the proof of the rule is as follows:

1. $\forall n \cdot (n \in \mathbb{N} \Rightarrow a \cap \overline{b} \cap v(n) \subseteq S(v'(n)) \cap \mathsf{grd}(S))$; from premises
2. $\forall n \cdot (n \in \mathbb{N} \Rightarrow a \cap v(n) \subseteq \mathcal{F}_m(b)(v'(n)))$; 1 and (4.12)
3. $a \subseteq \mathcal{F}_m(b)(a)$; premise and (4.12)
4. $a \subseteq \mathsf{fix}(\mathcal{F}_m(b))$; 3, 2, theorem 3
5. $a \mapsto b \in \mathcal{T}_m$; 4 and (4.13)
6. $a \mapsto b \in \mathcal{L}_{\mathcal{E}_m}$; 5, equality (4.15)

Antecedent of this rule corresponds to sufficient conditions in [7] to prove liveness properties and it is the only rule concerning the proof of liveness properties. Soundness of this rule is proved here in a more direct way.

Soundness of this rule is given without reasoning over state-traces, taking advantage of the fixpoint semantics approach.

4.4.3 From Weak Fairness to Minimal Progress

Using the variant theorem, we prove a sufficient condition to establish that a liveness property under minimal progress, follows from a corresponding property proved under weak fairness and from the decrement of a variant:

ANTECEDENT	CONSEQUENT
$\forall n \cdot (n \in \mathbb{N} \Rightarrow \overline{b} \cap v(n) \subseteq S(v'(n)))$ $a \mapsto b \in \mathcal{L}_{\mathcal{E}_w}$	$a \mapsto b \in \mathcal{L}_{\mathcal{E}_m}$

The proof of these conditions is given by the *Variant Theorem*. In order to apply the theorem, we need to identify an invariant set under $\mathcal{F}_m(b)$. However, as the sets a and b cannot be proved as invariants, we prove that the least fixpoint of $\mathcal{F}_w(b)$ is invariant under $\mathcal{F}_m(b)$, that is $\mathsf{fix}(\mathcal{F}_w(b)) \subseteq \mathcal{F}_m(b)(\mathsf{fix}(\mathcal{F}_w(b)))$. This proof requires the following lemma:

$$\forall \alpha \cdot (\mathcal{F}_w(b)^\alpha \subseteq b \cup (\mathsf{grd}(S) \cap S(\mathsf{fix}(\mathcal{F}_w(b))))) \tag{4.34}$$

The proof of (4.34) is done by transfinite induction; it is presented in appendix C.3.5. Using (4.34), the proof of sufficient conditions are as follows:

1. $\mathsf{fix}(\mathcal{F}_w(b)) \subseteq b \cup (\mathsf{grd}(S) \cap S(\mathsf{fix}(\mathcal{F}_w(b))))$; Theorem 1, (4.34)
2. $\mathsf{fix}(\mathcal{F}_w(b)) \subseteq \mathcal{F}_m(b)(\mathsf{fix}(\mathcal{F}_w(b)))$; 1 and (4.12)
3. $\forall n \cdot (n \in \mathbb{N} \Rightarrow \mathsf{fix}(\mathcal{F}_w(b)) \cap v(n) \subseteq b \cup \mathsf{grd}(S) \cap S(v'(n)))$; 2 and premise
4. $\forall n \cdot (n \in \mathbb{N} \Rightarrow \mathsf{fix}(\mathcal{F}_w(b)) \cap v(n) \subseteq \mathcal{F}_m(b)(v'(n)))$; 3 and (4.12)
5. $\mathsf{fix}(\mathcal{F}_w(b)) \subseteq \mathsf{fix}(\mathcal{F}_m(b))$; 4,2 and th. 3
6. $a \mapsto b \in \mathcal{T}_w$; premise, eq. (4.28)
7. $a \subseteq \mathsf{fix}(\mathcal{F}_w(b))$; 6 and def. $\mathcal{T}_w$
8. $a \subseteq \mathsf{fix}(\mathcal{F}_m(b))$; 7 and 5
9. $a \mapsto b \in \mathcal{L}_{\mathcal{E}_m}$; 8, def. $\mathcal{T}_m$, eq. (4.15)

Example 4.1

The example 3.7 proves that the producer-consumer system PC, specified in the example 2.15, satisfies the property $d \in P \leadsto d \in C$ under a weak fairness assumption, stating that any data d in the producer set, is eventually transferred to the consumer set.

The example 3.10 shows that the refinement *PC1* of the abstract system PC preserves the property

$$d \in P \leadsto d \in C_c$$

under a weak fairness assumption. If the invariant of *PC1* is strengthened with the equality $D = P \cup buf \cup C_c$, the lexicographic variant $[\mathsf{card}P, \mathsf{card}buf]$ is decremented by all events in *PC1* under the assumption $d \notin C_c$, that is:

$$d \notin C_c \wedge [\mathsf{card}P, \mathsf{card}buf] = [m, n] \Rightarrow [prod \ [\!] \ cons \ [\!] \ env_c] \ [\mathsf{card}P, \mathsf{card}buf] \prec [m, n]$$

holds in *PC1*. This fact, and the rule presented in this section, allows us to conclude that

$$d \in P \leadsto d \in C_c$$

holds in *PC1* under a minimal progress assumption.

□

4.5 Conclusion

In this chapter we presented the main theoretical results concerning the semantics of liveness properties. The semantics is presented in a set theoretical framework where events and iteration of events under minimal progress or weak fairness assumptions are modeled by set transformers. Liveness properties and the sets where termination of iteration is guaranteed are characterized by binary relations on subsets of states.

We start by introducing set transformers which model the weakest precondition and liberal weakest precondition of events. A special set transformer, named "dovetail", is presented to model a fair choice. Moreover, extreme solutions for equations with set transformers are defined as least and greatest fixpoints.

The semantics for *leadsto* is presented in a structured way. First the semantics is presented without concerns of fairness. The interpretation domain of *leadsto* properties is defined by a *termination relation* $\mathcal{T}$. Any pair of subsets of states $p \mapsto q$ in this relation denotes that the iteration of events, under a certain fairness assumption, starting in any state of p, terminates into a state in q. The opening operator $\widehat{\ }$ is used to model iteration of events. The body of the iteration is a guarded command $\mathcal{F}(q)$ with guard $\overline{q}$ and body W. W denotes the weakest precondition of a step in the iteration and it is not defined at this point; it is only stated that W must be monotonic and strict. The intended meaning of the iteration $\mathcal{F}(q)\widehat{\ }$ is that the system executes one or many steps W until it arrives into a state where q holds, and then the iteration terminates. Formally, the termination set of $\mathcal{F}(q)\widehat{\ }$ is given by the least fixpoint of the set transformer $\overline{q} \Longrightarrow W$. The *reachability relation* $\mathcal{L}_{\mathcal{E}}$ is defined as a binary relation on subsets of states in order to characterize general liveness properties. $\mathcal{L}_{\mathcal{E}}$ is inductively defined: it contains the base relation $\mathcal{E}$, it is transitive and disjunctive. The base relation characterizes basic liveness properties under some fairness assumption. At this time, $\mathcal{E}$ is not defined, only few properties relating $\mathcal{E}$ to the step W are required. After having defined the termination and reachability relations, the theorem asserting the equality between $\mathcal{L}_{\mathcal{E}}$ and $\mathcal{T}$ is stated. In another words, the theorem asserts that the rules used to derive *leadsto* properties are sound ($\mathcal{L}_{\mathcal{E}} \subseteq \mathcal{T}$) and relatively complete ($\mathcal{T} \subseteq \mathcal{L}_{\mathcal{E}}$).

The next step is the instantiation of the termination and reachability relation to consider minimal progress and weak fairness assumptions. For the case of minimal progress, the step of the iteration of events W_m is defined as preconditioned set transformer $\mathsf{grd}(S) \mid S$ which models the nondeterministic choice of events when an event is enabled. The basic relation under minimal progress assumptions $\mathcal{E}_m$ is defined as the pairs $\{x \mid I \wedge P\} \mapsto \{x \mid I \wedge Q\}$ where P and Q satisfy the MP0 and MP1 rules. These definitions satisfy the hypothesis of the soundness and completeness theorem which allow us to assert that the reachability and termination relation under minimal progress assumptions are equal ($\mathcal{L}_{\mathcal{E}_m} = \mathcal{T}_m$).

The instantiation of the termination relation to the weak fairness assumption needs the dovetail operator. Using this operator, a fair loop $Y(q)(G)$ is defined which models the iteration of the helpful event G under the weak fairness assumption. The weakest precondition of the step W_w to establish q is defined as the union, over all helpful events G in the system, of the weakest precondition of fair loops to establish q ($Y(q)(G)(q)$). The definition of the basic relation under weak fairness needs the definition of the basic relation $\mathcal{E}(G)$, for a helpful event G. It is defined as the pairs $\{x \mid I \wedge P\} \mapsto \{x \mid I \wedge Q\}$, where P and Q satisfy the WF0 and WF1 rules. The basic relation under weak fairness assumptions $\mathcal{E}_w$ is defined as the union, over all helpful events G in the system, of basic relations for helpful events $\mathcal{E}(G)$.These definitions satisfy the hypothesis of the soundness and completeness theorem which allow us

to assert that the reachability and termination relation under weak fairness assumptions are equal ($\mathcal{L}_{\mathcal{E}_w} = \mathcal{T}_w$).

The chapter terminates with a section illustrating the derivation of liveness properties as an example of the utility of our semantical work. The first example (subsection 4.4.2) shows how we relate the proof rules for modalities of Abrial and Mussat ([7]) to our *leadsto* properties. In fact, these rules are a particular case in our framework. The second example (subsection 4.4.3) is an attempt to provide a rule to verify the "implementation" of weak fairness assumptions in systems with minimal progress assumptions. The rule states that any *leadsto* property proved under a weak fairness assumption holds in a system with minimal progress assumptions if a variant is decremented by any event in the system.

The work reported in this chapter has been published as a research rapport in [58] and as a paper in [61].

Chapter 5

Soundness of Rules for Preserving Liveness Properties

Résumé

Dans ce chapitre, nous abordons la cohérence des règles permettant la préservation des propriétés de base, sous les hypothèses de progrès minimal et d'équité faible. Nous avons supposé dans le chapitre 3 que la préservation des propriétés de base est une condition suffisante pour de même la préservation des propriétés de vivacité générale. Dans ce chapitre, nous l´enonçons comme un théorème, que nous prouvons dans le cadre ensembliste introduit au chapitre 4. À partir de ce théorème, nous montrons la cohérence d'une part des règles BMP et LMP pour la préservation des propriétés de base sous l'hypothèse d'équité faible, et d'autre part, des règles SAP et LIP pour la préservation des propriétés de base sous l'hypothèse d'équité faible.

In section 3.4, rules for preserving basic liveness properties under refinement are proposed. We claim that the preservation of these properties guarantees the preservation of general liveness properties. In this chapter, a formal proof of this claim is presented and we prove soundness of the rules for preserving basic properties under minimal progress and weak fairness assumptions.

This chapter is divided in three sections. The first one presents general characteristics of refinements of B event systems in a set theoretical framework. Moreover, it proves that preserving basic properties is a sufficient condition for preservation of general liveness properties. Soundness of the rules for preserving basic liveness properties under minimal progress and weak fairness assumptions is proved in sections two and three respectively.

5.1 Refinement in a Set Theoretical Framework

In chapter 4 we consider a system $\mathcal{S}$ with state variable x, invariant I, set of events E and bounded choice of events in E denoted by S. In this chapter a refinement $\mathcal{T}$ of $\mathcal{S}$ is considered. This refinement is made up of a state variable y, gluing invariant J and set of events E' containing two disjoint subsets: the set of concrete events and the set of new events. Each concrete event F' of $\mathcal{T}$ is a refinement of an abstract event F in $\mathcal{S}$. New events refine *skip*.

In a set theoretical framework, the refinement $\mathcal{T}$ transforms a concrete state v which contains all values of the concrete variable satisfying the gluing invariant and related to abstract states:

$$v = \{y \mid \exists x \cdot (I(x) \wedge J(x, y))\}$$

In this way, any event in $\mathcal{T}$ can be modeled by a set transformer of type $\mathbb{P}(v) \rightarrow \mathbb{P}(v)$. The refinement relation r is defined as a total relation between the concrete state space and the abstract one which contains all pairs of values satisfying both the abstract and gluing invariants:

$$r = \{y \mapsto x \mid I(x) \wedge J(x, y)\}$$

For each abstract event F in the model $\mathcal{S}$ there exists a concrete event F' in the refinement $\mathcal{T}$ satisfying the following inclusion [2]:

$$\forall t \cdot (t \in \mathbb{P}(v) \Rightarrow F(\overline{r[\overline{t}]}) \subseteq \overline{r[\overline{F'(t)}]}) \tag{5.1}$$

From this inclusion follows that for any subset s of u, the inverse image under r of the weakest precondition for F to establish s is included in the weakest precondition for F' to establish the inverse image of s under r:

$$\forall s \cdot (s \in \mathbb{P}(u) \Rightarrow r^{-1}[F(s)] \subseteq F'(r^{-1}[s])) \tag{5.2}$$

The proof of this inclusion is based on equivalence $r[a] \subseteq b \equiv r^{-1}[\overline{b}] \subseteq \overline{a}$, where a and b are universally quantified over $\mathbb{P}(v)$ and $\mathbb{P}(u)$ respectively. The reference to this equivalence in the proof is given as "(Equ)". The proof of (5.2) is:

1.	$F(\overline{r[\overline{r^{-1}[s]}]}) \subseteq \overline{r[\overline{F'(r^{-1}[s])}]}$	; From (5.1)
2.	$r[\overline{F'(r^{-1}[s])}] \subseteq \overline{F(r[\overline{r^{-1}[s]}])}$	; 1
3.	$r^{-1}[F(r[\overline{r^{-1}[s]}])] \subseteq F'(r^{-1}[s])$	; 2 and Equ
4.	$r^{-1}[s] \subseteq r^{-1}[s]$	; trivial
5.	$r[\overline{r^{-1}[s]}] \subseteq \overline{s}$	; 4 and Equ
6.	$s \subseteq \overline{r[\overline{r^{-1}[s]}]}$	; 5
7.	$r^{-1}[F(s)] \subseteq r^{-1}[\overline{F(r[\overline{r^{-1}[s]}])}]$	; 6 and monotony
8.	$r^{-1}[F(s)] \subseteq F'(r^{-1}[s])$	; 7 and 3

□

Denoting by H the bounded choice of new events in $\mathcal{T}$ and considering that $skip(s) = s$, for any s in $\mathbb{P}(u)$, from (5.2) follows:

$$\forall s \cdot (s \in \mathbb{P}(u) \Rightarrow r^{-1}[s] \subseteq H(r^{-1}[s])) \tag{5.3}$$

which states that the new events do not modify concrete states, related by r to abstract ones.

Apart from safety properties specified as invariants in an event B model $\mathcal{S}$, we consider liveness properties specified by the *leads-to* relation. In a set theoretical framework, in chapter 4, we give an equivalent definition of the *leads-to* relation in term of the reachability relation $\mathcal{L}_{\mathcal{E}}$ stated in definition 4.4, which is a subset of $\mathbb{P}(u) \times \mathbb{P}(u)$. We recall that the base relation $\mathcal{E}$ in definition 4.4 must be instantiated to the basic relation $\mathcal{E}_m$ or $\mathcal{E}_w$ to get the definitions of

the reachability relation under minimal progress or weak fairness assumptions. In appendix A we prove that for any subsets p and q of u, containing states satisfying the predicates P and Q respectively, if the pair $p \mapsto q$ is contained in $\mathcal{L}_{\mathcal{E}}$, then it is equivalent to assert that the property $P \rightsquigarrow Q$ holds in $\mathcal{S}$.

In the refinement $\mathcal{T}$, the reachability relation is denoted by $\mathcal{L}_{\mathcal{E}'}$, which is a subset of $\mathbb{P}(v) \times \mathbb{P}(v)$. In $\mathcal{L}_{\mathcal{E}'}$, the base relation $\mathcal{E}'$ must be instantiated to the base relations under minimal progress ($\mathcal{E}'_m$) or weak fairness ($\mathcal{E}'_w$) to get the reachability relation under minimal progress or weak fairness assumptions in the refinement $\mathcal{T}$.

In section 3.4 we claim that preserving in $\mathcal{T}$ basic liveness properties proved in $\mathcal{S}$, is a sufficient condition to assert the preservation of general liveness properties. To express this claim in a more precise way, a subset β of $\mathcal{E}$ is considered. Taking the induction scheme of definition 4.4, the reachability relation $\mathcal{L}_\beta$ can be defined and it denotes all general liveness properties which can be generated from the set of basic properties β. From the closure clause in the definition of $\mathcal{L}_\beta$ and the inductive definition of $\mathcal{L}_{\mathcal{E}}$ follows that any property in $\mathcal{L}_\beta$ is as well a property of $\mathcal{L}_{\mathcal{E}}$, that is $\mathcal{L}_\beta \subseteq \mathcal{L}_{\mathcal{E}}$. Now we can state the following theorem:

Theorem 4 (Preservation of General Liveness Properties)
Let $\mathcal{S}$ be an abstract model, $\mathcal{T}$ a refinement of $\mathcal{S}$ with refinement relation r and β a subset of $\mathcal{E}$, the set of basic properties in $\mathcal{S}$. If any property in β is preserved in $\mathcal{T}$:

$$\forall (p, q) \cdot (p \mapsto q \in \beta \Rightarrow r^{-1}[p] \mapsto r^{-1}[q] \in \mathcal{L}_{\mathcal{E}'})$$

where $\mathcal{L}_{\mathcal{E}'}$ denotes the reachability relation in $\mathcal{T}$, then any general liveness property in $\mathcal{L}_\beta$ is preserved in $\mathcal{T}$:

$$\forall (p, q) \cdot (p \mapsto q \in \mathcal{L}_\beta \Rightarrow r^{-1}[p] \mapsto r^{-1}[q] \in \mathcal{L}_{\mathcal{E}'})$$

The proof of this theorem proceeds by induction on the structure of $\mathcal{L}_\beta$. For any p and q in $\mathbb{P}(u)$, let $P(p, q)$ be the property:

$$P(p, q) \equiv r^{-1}[p] \mapsto r^{-1}[q] \in \mathcal{L}_{\mathcal{E}'}$$

Three proofs are required:

- Base: If $p \mapsto q \in \beta$ then $P(p, q)$.
- Transitivity: If the pairs $o \mapsto p$ and $p \mapsto q$ are in $\mathcal{L}_\beta$ and they satisfy the inductive hypothesis $P(o, p)$ and $P(p, q)$, then $P(o, q)$ holds.
- Disjunction: If $l \times \{q\} \subseteq \mathcal{L}_\beta$ and $P(p, q)$ holds for any p in l, then $P(\bigcup(l), q)$ holds.

The base case follows from the premise stating the preservation of basic liveness properties. Transitivity and disjunction cases follow from the inductive definitions of $\mathcal{L}_\beta$ and $\mathcal{L}_{\mathcal{E}'}$.

□

The following sections show how the premise of theorem 4 is satisfied in the case of a minimal progress and weak fairness assumptions.

5.2 Preserving Liveness under Minimal Progress

The goal of this section is to prove that the BMP and LMP rules, stated in section 3.4.1 as sufficient conditions to preserve basic liveness properties under a minimal progress assumption, allow the proof of the premise of theorem 4, when the basic relation $\mathcal{E}$ is instantiated to the basic relation under minimal progress $\mathcal{E}_m$. The proof lies in the fact that refinement conditions and the BMP and LMP rules can be used to derive a basic liveness property holding in $\mathcal{T}$, and then the induction theorem IND can be used to prove the premise.

In the set theoretical framework, the BMP and LMP rules can be stated in an equivalent way as the following inclusions:

$$\forall n \cdot (n \in W \Rightarrow w(n) \subseteq H(w'(n))) \tag{5.4}$$

$$r^{-1}[\overline{S(\varnothing)}] \subseteq \overline{(T \,[\!]\, H)(\varnothing)} \tag{5.5}$$

where W is a well founded set and the functions w and w' are defined as follows:

$$w = \lambda n \cdot (n \in \mathbb{N} \mid \{\, z \mid z \in v \wedge V(z) = n \,\}) \tag{5.6}$$

$$w' = \lambda n \cdot (n \in \mathbb{N} \mid \{\, z \mid z \in v \wedge V(z) < n \,\}) \tag{5.7}$$

In appendices D.1.1 and D.1.1 can be found the proofs of the equivalence between the BMP and LMP rules and the inclusions (5.4) and (5.5).

In order to prove the hypothesis of the theorem 4 when the basic relation $\mathcal{E}$ is instantiated to the basic relation under a minimal progress assumption $\mathcal{E}_m$, and therefore $\beta \subseteq \mathcal{E}_m$, we take as assumption the antecedent of the implication in the hypothesis of the theorem

$$p \mapsto q \in \beta$$

and we prove

$$r^{-1}[p] \mapsto r^{-1}[q] \in \mathcal{L}_{\mathcal{E}'_m}$$

where $\mathcal{E}'_m$ denotes the basic relation under minimal progress in the refinement $\mathcal{T}$:

$$\mathcal{E}'_m = \{\, a \mapsto b \mid a \subseteq v \wedge b \subseteq v \wedge a \cap \overline{b} \subseteq (T \,[\!]\, H)(b) \cap \mathsf{grd}(T \,[\!]\, H) \,\} \tag{5.8}$$

In the proof, for the sake of simplicity, p' and q' denote the inverse image under r of the sets p and q respectively:

$$p' = r^{-1}[p] \qquad \text{and} \qquad q' = r^{-1}[q]$$

The proof needs the following inclusion which is proved in appendix D.1.3:

$$p' \cap \overline{q'} \subseteq r^{-1}[p \cap \overline{q}] \tag{5.9}$$

The proof of $p' \mapsto q' \in \mathcal{L}_{\mathcal{E}'_m}$ is as comes next.

From the antecedents $p \mapsto q \in \beta$ and $\beta \subseteq \mathcal{E}_m$ and the definition of $\mathcal{E}_m$ follows:

$$p \cap \overline{q} \subseteq S(q) \cap \mathsf{grd}(S) \tag{5.10}$$

The inclusion $r^{-1}[S(q)] \subseteq T(r^{-1}[q])$ follows by instantiating F to S, F' to T and s to q in (5.2). Weakening the postcondition of T in this last inclusion and using (5.10) and monotonicity of image we derive:

$$r^{-1}[p \cap \overline{q}] \subseteq T(q' \cup (r^{-1}[p \cap \overline{q}] \cap w'(n))) \tag{5.11}$$

Instantiating s to $p \cap \overline{q}$ in (5.3) allows the derivation of $r^{-1}[p \cap \overline{q}] \subseteq H(r^{-1}[p \cap \overline{q}])$. Weakening the conjunction of this last inclusion and (5.4) gives as outcome:

$$r^{-1}[p \cap \overline{q}] \cap w(n) \subseteq H((r^{-1}[p \cap \overline{q}] \cap w'(n)) \cup q') \tag{5.12}$$

Using the inclusion (5.9) and weakening the conjunction of (5.11) and (5.12) allows the derivation of

$$p' \cap \overline{q'} \cap w(n) \subseteq (T \,[\!]\, H)(p' \cap w'(n) \cup q') \tag{5.13}$$

On another hand, from (5.10) follows $r^{-1}[p \cap \overline{q}] \subseteq r^{-1}[\mathsf{grd}(S)]$. This last inclusion, (5.9) and the rule (5.5), by transitivity, allows the derivation of

$$p' \cap \overline{q'} \cap w(n) \subseteq \mathsf{grd}(T \,[\!]\, H) \tag{5.14}$$

The conjunction of (5.13) and (5.14) is a sufficient condition to guarantee, by the definition of $\mathcal{E}'_m$, that the pair $p' \cap w(n) \mapsto p' \cap w'(n) \cup q'$ belongs to $\mathcal{E}'_m$. By the equivalence (A.3) of appendix A, instantiated to the refined state space v, we derive:

$$y \in p' \cap w(n) \gg_m y \in p' \cap w'(n) \vee y \in q'$$

Considering that $y \in w(n)$ and $y \in w'(n)$, from definitions (5.6) and (5.7), are equivalent to $y \in v \wedge V = n$ and $y \in v \wedge V < n$ respectively, and by application of the BRL rule to the derived basic property we get:

$$y \in p' \wedge V = n \rightsquigarrow y \in p' \wedge V < n \vee y \in q'$$

Then, the property $y \in p' \rightsquigarrow y \in q'$ can be derived by the induction theorem IND of *leads-to*. Finally, from this *leads-to* property and the instantiation of the equivalence (4.7) to the refined state of $\mathcal{T}$, we derive the goal:

$$p' \mapsto q' \in \mathcal{L}_{\mathcal{E}'_m}$$

5.3 Preserving Liveness under Weak Fairness

The proof of the hypothesis of theorem 4, instantiating $\mathcal{E}$ to the basic relation under weak fairness assumption $\mathcal{E}_w$, follows the same steps as the previous proof under the minimal progress condition. That is, taking the antecedent of the hypothesis of theorem 4 as a local hypothesis $p \mapsto q \in \beta$, we derive the *leads-to* property $y \in r^{-1}[p] \rightsquigarrow y \in r^{-1}[q]$ holding in the refinement $\mathcal{T}$ and therefore, we prove the goal of theorem 4.

In this section, β is a subset of $\mathcal{E}_w$, the basic relation under weak fairness assumptions. As stated in (4.24), $\mathcal{E}_w$ is the union of basic relations for helpful events: $\mathcal{E}_w = \bigcup G{\cdot}(G \in E \mid \mathcal{E}(G))$. Therefore, from the antecedent of the hypothesis of theorem 4, $p \mapsto q \in \beta$, it follows that there exists a helpful event G in E such that:

$$p \mapsto q \in \mathcal{E}(G) \tag{5.15}$$

In the following paragraphs we prove that the LIP and SAP proof obligations are sufficient conditions to guarantee that (5.15) under the refinement conditions, implies

$$p' \mapsto q' \in \mathcal{L}_{\mathcal{E}'_w}$$

where p' and q' are, as in the previous section, the inverse image under r of the sets p and q respectively and $\mathcal{E}'_w$ denotes the basic relation under weak fairness assumptions in the refinement $\mathcal{T}$. As in the definition of $\mathcal{E}_w$ in $\mathcal{S}$, the basic relation $\mathcal{E}'_w$ is defined as the union of basic relations for helpful events in E', the set of events in the refinement $\mathcal{T}$:

$$\mathcal{E}'_w = \bigcup G' \cdot (G' \in E' \mid \mathcal{E}'(G'))$$

where $\mathcal{E}'(G')$ is the basic relation for any helpful event G' in E', defined like $\mathcal{E}(G)$ (4.24):

$$\mathcal{E}'(G') = \{\, a \mapsto b \mid a \subseteq v \wedge b \subseteq v \wedge a \cap \overline{b} \subseteq (T \,[\!]\, H)(a \cup b) \cap (G')(b) \cap \mathsf{grd}(G') \,\}$$

We recall that T is the choice of concrete events refining S and H are new events refining *skip*.

Following the notation in section 3.4.2, where S is considered as the choice of events F and G:

$$S = F \,[\!]\, G$$

with G denoting a helpful event and F the choice of events in $E - \{G\}$, the concrete events T in the refinement $\mathcal{T}$ is made up of the choice of events F' and G':

$$T = F' \,[\!]\, G'$$

where F' is the refinement of F and G' the refinement of G.

The proof uses the PSP theorem for the *leads-to* relation. This theorem is stated as follows in our set theoretical framework:

Theorem 5 (PSP Theorem under Weak Fairness)
Let $\mathcal{S}$ be any model with state space u, choice of events S and basic relation $\mathcal{E}_w$. Let p, q, a and b be any subset of u such that the following holds:

$$p \mapsto q \in \mathcal{L}_{\mathcal{E}_w} \qquad \textit{and} \qquad a \cap \overline{b} \subseteq S(a \cup b)$$

Then it follows:

$$p \cap a \mapsto q \cap a \cup b \in \mathcal{L}_{\mathcal{E}_w}$$

This theorem is proved in appendix D.2.1 by induction on the structure of $\mathcal{L}_{\mathcal{E}_w}$.

Considering the sets p and q in (5.15) as the subsets of states in u satisfying the predicates P and Q respectively:

$$p = \{\, z \mid z \in u \wedge P(z) \,\} \qquad \text{and} \qquad q = \{\, z \mid z \in u \wedge Q(z) \,\}$$

the LIP and SAP proof obligations can be stated in an equivalent form in our set theoretical framework:

$$r^{-1}[p \cap \overline{q}] \cap \mathsf{grd}(G') \subseteq (F' \,[\!]\, H)(\mathsf{grd}(G')) \tag{5.16}$$

$$r^{-1}[p \cap \overline{q}] \cap \overline{\mathsf{grd}(G')} \mapsto \mathsf{grd}(G') \in \mathcal{L}_{\mathcal{E}'_w} \tag{5.17}$$

The proof of these equivalences is given in appendices D.2.2 and D.2.3. The proof of the goal $p' \mapsto q' \in \mathcal{L}_{\mathcal{E}'_w}$ is as comes next.

From (5.15) and definition of $\mathcal{E}(G)$ follows:

$$p \cap \overline{q} \subseteq S(p \cup q) \cap G(q) \cap \mathsf{grd}(G) \tag{5.18}$$

Considering S as the choice $F \parallel G$, from (5.18), by monotonicity of the image operator, we derive:

$$r^{-1}[p \cap \overline{q}] \subseteq r^{-1}[F(p \cup q)] \cap r^{-1}[G(q)] \tag{5.19}$$

As F' is a refinement of F and G' a refinement of G, by application of (5.2) we get $r^{-1}[F(p \cup q)] \subseteq F'(r^{-1}[p \cup q])$ and $r^{-1}[G(q)] \subseteq G'(r^{-1}[q])$. Therefore, from these last derivations and (5.19) we conclude by transitivity:

$$r^{-1}[p \cap \overline{q}] \subseteq F'(p' \cup q') \cap G'(q') \tag{5.20}$$

From (5.3) we conclude $r^{-1}[p \cap \overline{q}] \subseteq H(r^{-1}[p \cap \overline{q}])$. By weakening the conjunction between this last inclusion and (5.16) we derive: $r^{-1}[p \cap \overline{q}] \cap \mathsf{grd}(G') \subseteq H(p' \cap \mathsf{grd}(G'))$. By weakening the conjunction of this inclusion and (5.20) we obtain:

$$r^{-1}[p \cap \overline{q}] \cap \mathsf{grd}(G') \subseteq (F' \parallel H)(p' \cap \mathsf{grd}(G') \cup q') \tag{5.21}$$

Weakening (5.20) allows the derivation of the following inclusion:

$$r^{-1}[p \cap \overline{q}] \subseteq G'(p' \cap \mathsf{grd}(G') \cup q') \tag{5.22}$$

The conjunction of (5.22) with (5.21), considering that $T = F' \parallel G'$, gives as result:

$$r^{-1}[p \cap \overline{q}] \cap \mathsf{grd}(G') \subseteq (T \parallel H)(p' \cap \mathsf{grd}(G') \cup q') \tag{5.23}$$

From (5.20) follows

$$r^{-1}[p \cap \overline{q}] \cap \mathsf{grd}(G') \subseteq G'(q') \cap \mathsf{grd}(G') \tag{5.24}$$

Considering the definition of $\mathcal{E}'(G')$ above and the inclusion (5.9), the conjunction of (5.24) and (5.23) allows us to conclude $p' \cap \mathsf{grd}(G') \mapsto q' \in \mathcal{E}'(G')$ and therefore

$$p' \cap \mathsf{grd}(G') \mapsto q' \in \mathcal{L}_{\mathcal{E}'_w} \tag{5.25}$$

Let a and b be the following sets:

$$a = p' \cap \overline{q'} \cap \overline{\mathsf{grd}(G')}$$
$$b = p' \cap \overline{q'} \cap \mathsf{grd}(G') \cup q'$$

By remarking that $a \cup b$ is equal to $p' \cup q'$ and by application of (5.9), from (5.20), (5.22) and (5.3) we conclude:

$$a \cap \overline{b} \subseteq (T \parallel H)(a \cup b) \tag{5.26}$$

Applying the PSP theorem to (5.17) and (5.26) we get

$$p' \cap \overline{q'} \cap \overline{\mathsf{grd}(G')} \mapsto p' \cap \mathsf{grd}(G') \cup q' \in \mathcal{L}_{\mathcal{E}'_w} \tag{5.27}$$

The property $q' \mapsto q' \in \mathcal{L}_{\mathcal{E}'_w}$ holds trivially. Applying SDR rule to this last property and (5.25) allows us to conclude:

$$p' \cap \mathsf{grd}(G') \cup q' \mapsto q' \in \mathcal{L}_{\mathcal{E}'_w} \tag{5.28}$$

From (5.28) and (5.27) by transitivity follows:

$$p' \cap \overline{q'} \cap \overline{\mathsf{grd}(G')} \mapsto q' \in \mathcal{L}_{\mathcal{E}'_w} \tag{5.29}$$

From (5.25) follows $p' \cap \overline{q'} \cap \mathsf{grd}(G') \mapsto q' \in \mathcal{L}_{\mathcal{E}'_w}$. Applying the disjunction rule to this property and (5.29) gives as result:

$$p' \cap \overline{q'} \mapsto q' \in \mathcal{L}_{\mathcal{E}'_w} \tag{5.30}$$

The property $p' \cap q' \mapsto q' \in \mathcal{L}_{\mathcal{E}'_w}$ holds trivially. Finally, the goal $p' \mapsto q' \in \mathcal{L}_{\mathcal{E}'_w}$ follows from this last derivation and (5.30) by application of the disjunction rule.

□

5.4 Conclusion

In this chapter we prove soundness of the rules for liveness preservation under refinement. The proof is carried out in the set theoretical framework of the previous chapter. Soundness of these rules is founded on the fact that preserving basic liveness properties guarantees the preservation of general liveness properties.

We consider a refinement $\mathcal{T}$ of an abstract system $\mathcal{S}$. The state variable of the refinement is denoted by y and its state space by v, which contains those concrete states related to the abstract ones by the refinement relation r. Under this framework, we state that a property $p \mapsto q \in \mathcal{L}_{\mathcal{E}}$ of the abstract system $\mathcal{S}$, is preserved in the refined system $\mathcal{T}$, if the property $r^{-1}[p] \mapsto r^{-1}[q] \in \mathcal{L}_{\mathcal{E}'}$ holds, where $\mathcal{E}'$ is a binary relation of subsets of v and $\mathcal{L}_{\mathcal{E}'}$ the reachability relation in the refinement. Moreover, a basic liveness property $p \mapsto q \in \mathcal{E}$ is preserved in $\mathcal{T}$, if the property $r^{-1}[p] \mapsto r^{-1}[q] \in \mathcal{L}_{\mathcal{E}'}$ holds in the refined system.

Theorem 4 states that the preservation of general liveness properties depends on the preservation of basic liveness properties. That is, considering any subset β of basic properties, $\beta \subseteq \mathcal{E}$, if all the basic properties in β are preserved, then any general liveness property generated from β, is preserved in the refinement. Therefore, soundness of rules preserving general liveness properties, is reduced to prove soundness of rules for the preservation of basic liveness properties.

Soundness proofs for basic liveness properties take advantage of the refinement relation between abstract and concrete events and between *skip* and new events. From these relations, and the proof obligations BMP and LMP for minimal progress, or LIP and SAP for weak fairness assumptions, we prove that basic liveness properties are preserved in the refinement.

We stress the fact that our proofs for preserving liveness properties are carried out by reasoning in the reachability relation only. We take advantage of the inductive definition of that relation. From Theorem 2, which states the equality between the reachability and termination relations, another kind of proof could be done, reasoning on the least fixpoints of set transformers. However, this kind of proofs seems to us more complicated. In order to argument our claim, we can compare our proof for the preservation of basic liveness properties under minimal progress assumption and the proof for liveness preservation in [7].

In [59] we reported the first version of the soundness proof for the LIP and SAP rules and the soundness proofs for BMP and LMP were published in [17].

Chapter 6

An example: Mutual Exclusion

Résumé

Dans ce chapitre, nous faisons une étude de cas qui montre notre approche pour la construction d'algorithmes. L'objectif principal de cette étude de cas est de montrer la spécification et la preuve des propriétés de vivacité dans le style UNITY *sous l'hypothèse d'équité faible. Nous commençons par la spécification d'un modèle abstrait ayant un haut niveau de non déterminisme et nous prouvons ses propriétés de sûreté et de vivacité. Ensuite, nous procédons au raffinement de ce modèle afin d'aboutir à deux types de mises en œuvre : une centralisée et une autre distribuée. La mise en œuvre centralisée modélise un serveur qui donne accès selon un ordre* fifo*. La mise en œuvre distribuée conduit à l'algorithme d'exclusion mutuelle de Ricart et Agrawala selon lequel les processus coopèrent pour accéder à la section critique.*

In this chapter, the approach to the development of systems under weak fairness assumptions is illustrated by two examples. These examples implement an algorithm to manage the mutual exclusion problem, under two supposed environments: a centralized and a distributed one. First, an abstract model is presented, which gives a highly non deterministic solution to the mutual exclusion problem. Two main properties are proved in this model: safety and liveness. The safety property indicates that there is at most, one process in the critical section. The liveness property states that any process requesting access to the critical section, eventually gets it. The first example refines the abstract model in an algorithm which implements a first input first output policy in the access to the critical section. The management of this policy is made by a server, which grants access rights to requesting processes. On the other hand, the second example refines the abstract model in a distributed algorithm, where each process is aware of requests made by other processes. Access to the critical section is given to a process with the oldest request. As these examples are refinements of the abstract model, satisfying the proof obligations for safety and liveness preservation, the two main properties of the abstract model are preserved in the centralized and distributed implementations of the mutual exclusion algorithm. The abstract model, its refinements and proof obligations for basic liveness properties have been entirely verified with the "Click'n Prove" prover [4].

This chapter is structured in three sections. The first section presents the abstract model of the algorithm. The second one shows two refinements where the first input first output policy is implemented and the third section illustrates four refinements that give the distributed implementation of the mutual exclusion algorithm.

6.1 The Abstract Model

The abstract model, named *Mutex*, considers a set of processes PR with cardinality greater than one. Each process can be in one of three states: *idle*, *waiting* or *active*. A process in state *idle* is out of the critical section, and it has no pending request to enter the critical section. When a process requests access, it passes to state *waiting*. Finally, when a process gets access to the critical section, its state becomes *active*. In the initial state of the system, the state of any process is *idle*. In order to decide which process gets access to the critical section, a logical clock is associated with each process. A logical clock is a variable of type $\mathbb{N}$, and its value is referenced as logical time, or simply time; the initial value of the clock, for any process, is zero. When a process asks for access to the critical section, the clock is updated; the new value of the clock must be greater than its current time. In this model, any process in state *waiting* with the lowest time in its clock, can be chosen to enter the critical section. When a process leaves the critical section, its state becomes *idle* again.

The formal definition of the abstract model needs variables st and t with the following type:

$$st \in PR \to ST,$$
$$t \in PR \to \mathbb{N}$$

where $ST = \{ID, WT, AC\}$, which denotes the set of states: *idle* (ID), *waiting* (WT) or *active* (AC). st assigns a state to each process, and t associates a logical clock with each process. In the initial state of the abstract model, all processes have an *idle* state, and its time stamp is zero.

For the sake of simplicity, the sets of processes in state *idle* (Idl), *waiting* (Wtg) and *active* (Act) are defined by the relational image:

$$Idl = st^{-1}[\{ID\}], \quad Wtg = st^{-1}[\{WT\}] \quad \text{and} \quad Act = st^{-1}[\{AC\}]$$

The main safety property in this model states that there is at most, one process in the critical section at any time. Formally it is specified by the following predicate

$$\mathsf{card}(Act) \leq 1$$

which is invariant in the abstract model.

Events in the abstract model are presented in figure 6.1; they code the informal behavior presented above. Event *req* (*request*) can be executed by any process p in state *idle*. When it is executed, the state of the process changes to *waiting*, and its new time becomes n, where n is greater than the current value of the clock $t(p)$. Event *ent* (*enter*) can be executed when the set of processes Wtg is not empty, and the critical section is not busy. Any process p, whose state is *waiting*, and its time is one of the lowest among the process in state *waiting*, can access the critical section if it is empty. When a process p leaves the critical section, event *rel* (*release*) is executed, and the state of p changes to *idle*.

The abstract model is a highly non deterministic specification to the mutual exclusion problem, and therefore, it is a unfair solution for processes waiting for the critical section. The cause for this nondeterminism, is that t is only a total function, and the set of processes with the lowest time m, equal to $\mathsf{min}(t[Wtg])$, is not necessary a singleton. Therefore, any process with time m can be chosen to enter the critical section, when event *ent* is executed. Process with time m which are not chosen, must wait another execution of event *ent*, to

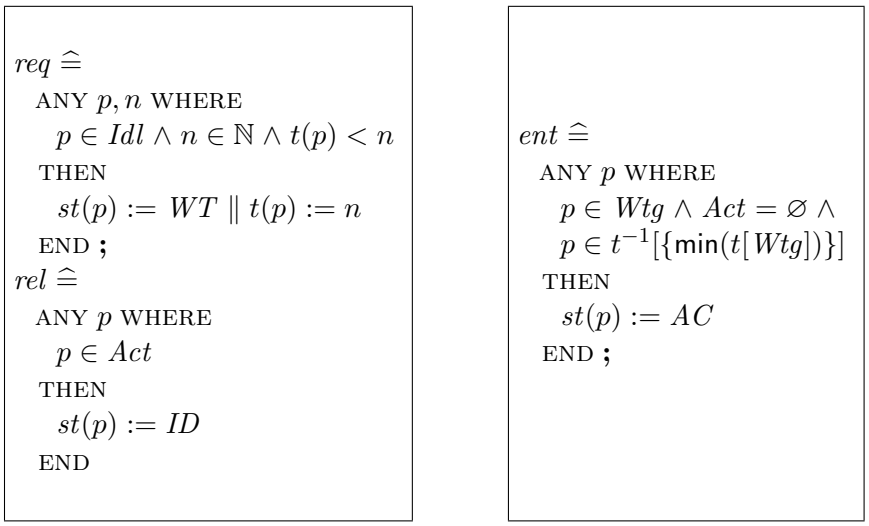

Figure 6.1: The Abstract Model: *Mutex*

possibly enter into the critical section. However, execution of event *req*, can add a new process to the set Wtg, with a time n lower than m. In this new state of the system, another execution of event *ent*, chooses the process with time n, and the processes with time m must wait again. From the fact that clocks are incremented each time that a process requests access to the critical section, any process in state *waiting* eventually accesses the critical section. This informal reasoning is formalized in the next paragraphs.

Liveness

The main liveness property of this algorithm is stated as follows: any process q from PR, requesting access to the critical section, eventually gets it. Formally this property is specified in the following way:

$$q \in Wtg \rightsquigarrow q \in Act \tag{6.1}$$

The main idea in the proof of this property is the notion of "distance". For any two processes o and p from PR, the value $t(o) - t(p)$ denotes the distance between these processes. As event *ent* grants access to one process with lower time, only the distances between q and any other process with lower time is of interest. When a process requests access, the sum of these distances is not incremented. However, that sum is decremented when a process goes into the critical section. It ensures that any requesting process goes eventually into the critical section. In order to formalize this reasoning, the following definitions are given:

$$lt(q) = \{\, z \mid z \in PR \wedge t(z) < t(q) \,\} \tag{6.2}$$

$$V1 = \Sigma z \cdot (z \in lt(q) \mid t(q) - t(z)) \tag{6.3}$$

$$V2 = \Sigma z \cdot (z \in lt(q) \cap Wtg \mid t(q) - t(z)) + \mathsf{card}(Wtg \cap t^{-1}[\{t(q)\}]) \tag{6.4}$$

$$V3 = \mathsf{card}(Act) \tag{6.5}$$

$lt(q)$ denotes the set of processes with a time lower than $t(q)$. *V1* represents the sum of distances between q and p, for any p with time lower than $t(q)$. *V2* stands for the sum of

two quantities: the sum of distances between q and requesting processes with lower time, and the number of requesting processes with time equal to $t(q)$. Finally, *V3* is the number of processes in the critical section.

When event *req* is executed, if the time of p is lower than $t(q)$, *V1* is decremented. If $t(q) \leq t(p)$, the values of *V1* and the sum of distances from processes in *Wtg* do not change. On another hand, the number of processes in *Wtg*, with time equal to $t(q)$, stays constant. Thus, the global variant is not incremented. When event *ent* is executed, the clocks are not modified, and *V1* does not change. If q does not changes to the *active* state, and $t(p) < t(q)$, the sum of distances from processes in *Wtg* decrements. If $t(p) = t(q)$ the number of processes waiting for the critical section decreases. Therefore, the global variant is decremented by the execution of *ent* or q goes into the critical section. Finally, execution of *rel* decrements the number of processes in the critical section. These arguments, allow the proof of the following basic liveness properties:

$$ent \cdot q \in Wtg \wedge Act = \varnothing \wedge V = v \gg (q \in Wtg \wedge V \prec v) \vee q \in Act \tag{6.6}$$

$$rel \cdot q \in Wtg \wedge Act \neq \varnothing \wedge V = v \gg q \in Wtg \wedge V \prec v \tag{6.7}$$

where the variant V is defined as follows:

$$V = [V1, V2, V3] \tag{6.8}$$

and v denotes the list $[k, l, m]$, for k, l and m universally quantified over the natural numbers, and $\prec$ is the *lower than* relation over a lexicographic order. Property (6.6) states that execution of *ent* decrements the variant while q is waiting for the critical section, or q goes into the critical section. Moreover, it states that any other event does not increment the variant. Property (6.7) specifies that execution of *rel* decrements the variant maintaining q in state *waiting*, while *req* and *ent* do not increment it. These properties are proved with proof obligations WF0 and WF1 presented in section 3.3.

Applications of the basic rule BRL to (6.6) and (6.7), and then disjunction rule DSJ, give the property:

$$q \in Wtg \wedge [V1, V2, V3] = [k, l, m] \rightsquigarrow (q \in Wtg \wedge [V1, V2, V3] \prec [k, l, m]) \vee q \in Act$$

Finally, application of the induction rule IND allows derivation of the goal: $q \in Wtg \rightsquigarrow q \in Act$.

Events in the abstract model represent a closed system with two entities: the process asking for access to the critical section, and the access mechanism. Events which model the request (*req*), and the release (*rel*) of the critical section, are executed when a process asks for access, or leaves the critical section. However, when the access mechanism decides that a waiting process must enter the critical section, event *ent* is executed. The validity of the main liveness property (6.1), as described by properties (6.6) and (6.7), needs to execute events *ent* and *rel* under a weak fairness assumption, however, there is not a weak fairness assumption about the execution of event *req*. This means that access to the critical section, for any process in state *waiting*, depends only on the weak fair execution of events *ent* and *rel*. Moreover, property (6.6) specifies the main liveness property which must be satisfied by the access mechanism, while (6.7) specifies the liveness property guaranteed by processes. The algorithms derived in this section, give two possible implementations of the access mechanism, whose main liveness property, is specified by property (6.6). Of course, preservation

of property (6.7) must be proved, in order to claim that the main liveness property of the abstract model holds in the implementations. On the other hand, as safety properties, specified by invariants in the model, are preserved by the very definition of refinement, in this paper only the preservation of liveness properties is considered.

Proof Obligations for *Mutex*

In this paragraph we show the number of proof obligations generated by the "Click'n Prove" prover in the verification of the *Mutex* model. The number of proofs is presented by a table containing four columns. The columns are labeled as follows:

NbObv Number of obvious proof obligations. They are simple proofs which are automatically removed by the proof obligations generator.

NbPO Number of no obvious proof obligations.

NbPRi Number of proof obligations proved with the interactive prover.

NbPRa Number of proof obligations proved with the automatic prover.

The rows of the table are labeled as follows:

Init Number of proof obligations generated by the initialization in the model.

events One row for each event in the model shows the number of proof obligations generated by the event.

Total Model The sum of proof obligations in rows "Lemmas", "Init" and "events".

Total Liveness The number of assertions generated by the WF0 and WF1 rules, required in the proof of the basic properties (6.6) and (6.7).

Total The sum of proof obligations in rows "Total Model" and "Total Liveness".

The table containing the number of proof obligations for the model *Mutex* is the following:

	NbObv	NbP0	NbPRi	NbPRa
Init	0	5	0	5
req	1	5	0	5
ent	3	3	0	3
rel	3	3	0	3
Total Model	7	16	0	16
Total Liveness	2	9	7	2
Total	9	25	7	18

6.2 A Centralized Implementation

In this section, a centralized algorithm is refined from the abstract model. The derivation is made by two refinements. In the first refinement, an infinite sequence is introduced which records the order of access requests, made by the processes. The sequence allows implementation of a first input first output policy in the access to the critical section. The second refinement introduces a new event which models a server. The server centralizes the requests, and assigns a token to the process which generated the oldest request; the token grants access to the critical section. The two refinements are presented in the following paragraphs.

6.2.1 A First Input First Output Policy

The main concern of this refinement, named *Fifo_1*, is the reduction of nondeterminism in the access mechanism. So, the total function, which models logical clocks for processes in the abstract model, is refined by a infinite sequence of processes, which gives basically the same information, but in a deterministic way. When a process asks for access to the critical section, it is simply appended to the sequence. A new variable is introduced, which is used as a pointer to the oldest request in the sequence. In this way, when the set of processes in state *waiting* is not empty, and the critical section is free, the process in the sequence, referenced by the pointer, can access the critical section.

The sequence and the pointer are denoted by two new variables qu and i, defined as follows:

$$qu \in \mathsf{seq}(PR),$$
$$i \in \mathbb{N}$$

In the initial state, qu is empty, and i is zero. The gluing invariant, which relates the function t of the abstract model with the new variable, is given by the following relation:

$$t = (\overline{\mathsf{ran}(qu)} \times \{0\}) \cup (qu^{-1} \mathbin{;} M(qu))$$

where $\overline{\mathsf{ran}(qu)}$ is equal to $PR - \mathsf{ran}(qu)$, and $M(qu)$ is a function of type $\mathbb{N}_1 \rightarrow \mathbb{N}$, defined as follows, for any non empty sequence se of processes, and any natural number n greater than zero:

$$M(se)(n) = \mathsf{max}(se^{-1}[\{se(n)\}])$$

$M(se)(n)$ denotes the greatest position in se of the process $se(n)$. In fact, from the definitions follows that $(qu^{-1} \mathbin{;} M(qu))$ is a partial function of type $PR \rightarrow \mathbb{N}_1$. The gluing invariant states that any process that is not in the sequence has time zero, and that the time of any process in the position n of the sequence is equal to $\mathsf{max}(qu^{-1}[\{qu(n)\}])$.

The proof of the gluing invariant needs the following invariants:

$$i \leq \mathsf{size}(qu),$$
$$st^{-1}[\{WT\}] = qu[i+1..\mathsf{size}(qu)],$$
$$(i+1..\mathsf{size}(qu) \lhd qu)^{-1} \in PR \mathrel{+\!\!\!\rightarrow} i+1..\mathsf{size}(qu)$$

The first invariant indicates that the pointer i is not greater than the size of qu, the second one states that Wtg, the set of processes in state *waiting*, is equal to the set of processes

in the interval $i+1..size(qu)$ of the sequence qu, and the third invariant asserts that qu, in the interval $i+1..size(qu)$, is an injective sequence, that is, its inverse is a partial function from PR to $i+1..\mathsf{size}(qu)$. From this last invariant follows that, any process cannot make a request, if its previous one is not served.

Events for system *Fifo_1* are presented in figure 6.2. Event *rel* remains the same as the abstract model one. Events *req* and *ent* are simplified, because the updates of logical clocks are not used any more. When a process asks for access to the critical section, it is simply appended to end of the sequence. When the pointer i is lower than the size of qu, which means that $Wtg \neq \varnothing$, and the critical section is empty, the process, pointed by $i+1$ in the sequence, accesses the critical section, and the pointer is updated to its new value $i+1$.

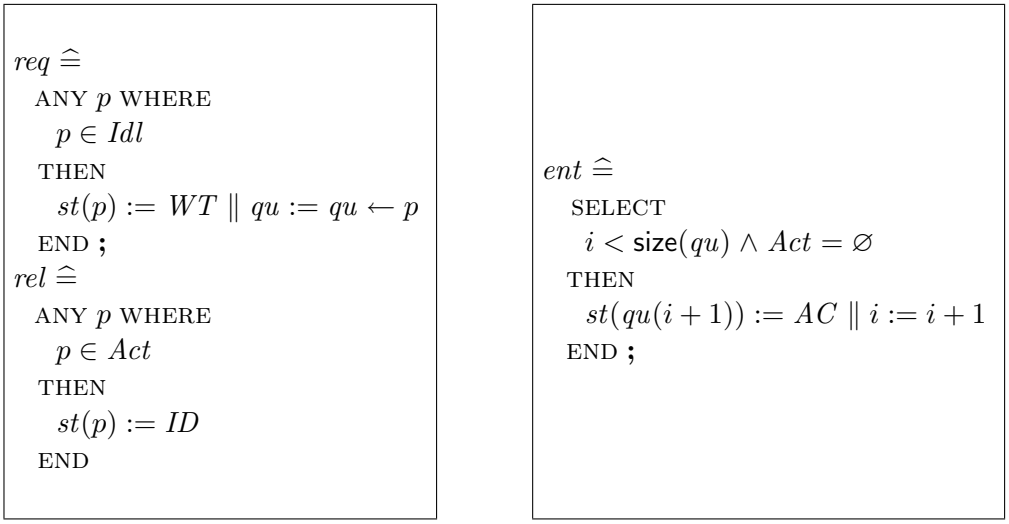

Figure 6.2: System *Fifo_1*

Preserving Liveness

This refinement must guarantee the preservation of properties (6.6) and (6.7), in order to claim that the main property (6.1) is preserved, as stated in section 3.4. However, by rule R1, preservation of (6.7) holds trivially, because the guard of event *rel* does not change in this refinement, so $Act \neq \varnothing \Rightarrow grd(rel)$. As we are interested in the development of the access mechanism, from now on, the guard of *rel* is not modified in the next refinements, which guarantees preservation of (6.7).

On another hand, from $q \in Wtg \wedge Act = \varnothing$ and the invariant $Wtg = qu[i+1..\mathsf{size}(qu)]$ follows $i < \mathsf{size}(qu) \wedge Act = \varnothing$, which is equivalent to the guard of event *ent* in this refinement. Therefore by the rule R1, the preservation of the basic property (6.6) holds.

The trivial proofs in the preservation of the main liveness property, comes from the fact that qu gives one of the possible orders, of processes in the set Wtg, accepted by the abstract model. However, in *Fifo_1*, the access mechanism satisfying liveness property (6.6), is not yet developed. This issue is treated in the next refinement.

Proof Obligations for *Fifo_1*

As in the abstract model, we present a table containing the number of proof obligations

generated by the refinement *Fifo_1*. The information in the table is interpreted as the table of the *Mutex* model. The row labeled "Lemmas" contains the number of proof obligations generated by some lemmas required in this refinement. The row labeled "Total Liveness" indicates the proof of the implication proving rule R1 in the preservation of (6.6). The table is as follows:

	NbObv	NbP0	NbPRi	NbPRa
Lemmas	0	3	3	0
Init	4	5	0	5
req	4	5	3	2
ent	6	5	3	2
rel	8	2	2	0
Total Refinement	22	20	11	9
Total Liveness	1	1	1	0
Total	23	21	12	9

6.2.2 A Token to Grant Access Rights

In this refinement, named *Fifo_2*, the access mechanism for the first input first output policy is developed. The infinite sequence in the previous refinement, is refined by an injective sequence, which only records the order of processes in state *waiting*. In this way, the access mechanism assigns a token to the first process in the sequence, which allows access to the critical section to that process.

In *Fifo_2*, the sequence qu and the pointer i of the previous refinement disappear. These variables are refined by a sequence sq, which records the order of requests. A new variable tk is introduced to manage the assignment of the token. The type of these variables is as follows:

$$sq \in \mathsf{iseq}(PR),$$
$$tk \in PR \to BOOL$$

For any process p, $tk(p)$ holds if the token is assigned to p. In the initial state of the system, $tk(p)$ is *false* for any process p, and the sequence sq is empty. The gluing invariant, relating the sequence sq with the variables qu and i of refinement *Fifo_1*, is as comes next:

$$sq = (a(i) \mathbin{;} (i + 1..\mathsf{size}(qu) \lhd qu))$$

where $a(n)$, for $n \in \mathbb{N}$, is a total function of type $\mathbb{N} \to \mathbb{N}$, defined for any $x \in \mathbb{N}$ as the sum $x + n$. This relation states that the process in the position j of sq, is the same as the process in the position $j + i$ of qu.

The following invariants are needed in the proof of the gluing invariant, which state properties in the management of the token:

$$\mathsf{card}(Tk) \leq 1,$$
$$Wtg \cap Tk \subseteq \{\mathsf{first}(sq)\},$$
$$Act \subseteq Tk,$$
$$Tk \subseteq \overline{Idl}$$

where $Tk = tk^{-1}[\{true\}]$. The first invariant states that at most, one process can have assigned the token. The second invariant indicates that any process in state *waiting*, which has assigned the token, is the first process in the sequence. The third invariant specifies that any process in state *active*, has assigned the token. Finally, the fourth invariant indicates that a process in state *idle* does not have assigned the token.

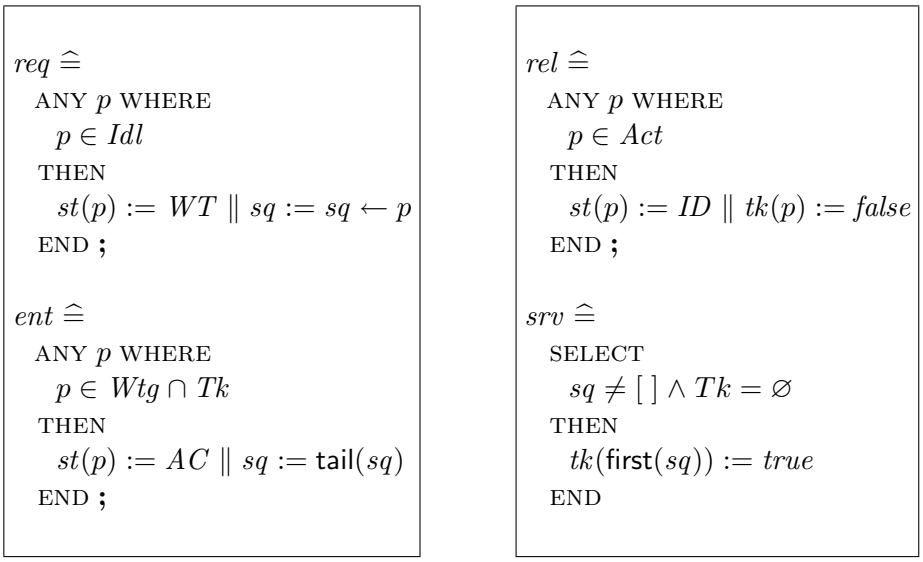

Figure 6.3: System *Fifo_2*

Events in the system *Fifo_2* are presented in figure 6.3. A new event *srv* (*server*) is introduced. It models a server which implements the access mechanism. When the sequence *sq* is not empty, and the token is not assigned to any process, the execution of *srv* assigns the token to the first process in the sequence. In this way, any process in state *waiting*, that has the token assigned, can access the critical section, as specified by event *ent*. Events *req* and *rel* have not practically changed with respect to system *Fifo_1*; in event *req*, only *qu* is replaced by *sq*, and in event *rel*, the release of the token is added.

Preserving Liveness

In the previous refinement, property (6.6), needed in the proof of the main liveness property, is trivially preserved. In this refinement, the access mechanism is introduced, and we prove that this mechanism and the processes preserve property (6.6).

Preserving property (6.6), requires the demonstration of the following properties, which are sufficient to prove LIP and SAP proof obligations:

$$q \in Wtg \wedge Act = \varnothing \wedge \neg grd(ent_2) \rightsquigarrow grd(ent_2) \tag{6.9}$$

$$q \in Wtg \wedge Act = \varnothing \wedge grd(ent_2) \Rightarrow [e]\, grd(ent_2) \tag{6.10}$$

where $grd(ent_2)$ is the guard of event *ent* in system *Fifo_2*, and e is universally quantified over the set of events $\{req, rel, srv\}$.

In fact, considering that the guard of event *ent* is equivalent to $Wtg \cap Tk \neq \varnothing$, property (6.9) follows by application of the BRL rule to the following basic property:

$$srv \cdot q \in Wtg \wedge Act = \varnothing \wedge Wtg \cap Tk = \varnothing \gg Wtg \cap Tk \neq \varnothing \tag{6.11}$$

This property specifies that the execution of event *srv* ensures that the token is assigned to a waiting process when *srv* is executed in a state where q is waiting for the critical section, the critical section is empty and the token is not assigned. In other words, this property simply specifies that execution of the access mechanism guarantees that the guard of event *ent* becomes enabled.

(6.11) is proved by the WF0 and WF1 proof obligations. On the other hand, (6.10) is easily demonstrated by weakest precondition and predicate calculus.

The sequence *sq*, shared between the event *req*, which models access requests to the critical section, and event *srv*, which models the server, can be refined in an abstract communication channel. In this way, a new event, that models a communication system, can be introduced to manage communications between the server and client processes. This approach has been taken in [60].

In the next example, a distributed implementation of the mutual exclusion problem, the refinement of abstract structures in communication channels is showed.

Proof Obligations for *Fifo_2*
In this table, we present the number of proof obligations for this refinement. The row labeled "Total Liveness" indicates the number of assertions generated by the WF0 and WF1 rules, needed in the proof of the property (6.11) and the assertions generated by the SAP rule (6.10). The table is as follows:

	NbObv	NbP0	NbPRi	NbPRa
Lemmas	0	2	2	0
Init	2	11	0	11
req	5	8	5	3
ent	5	11	7	4
rel	5	9	5	4
srv	9	6	4	2
Total Refinement	26	47	23	24
Total Liveness	1	10	5	5
Total	27	57	28	29

6.3 A Distributed Implementation

In this section, a distributed solution to the mutual exclusion problem, known as the Ricart-Agrawala algorithm [55], is derived from the abstract model. Informally, this algorithm is explained as follows. A process in state *idle*, asking for access to the critical section, sends a request message to all the processes in the system, and then it passes to state *waiting*. The request message contains the identifier of the process which sends the message, and a time stamp, which corresponds to the time of its local clock when the request is generated. The couples made up of time stamp and identifier of process, form a total order that is used to decide which process accesses the critical section. Any process in state *waiting*, which receives a reply message from all processes in the system, can access the critical section. When a process p receives a request message, it compares its local clock and the time stamp in the message; the maximum of these values is recorded to update its local clock when a new request is generated. The next action of the process, depends on its state and the outcome

of the comparison. If p is the state *idle*, or its clock and its identifier are, in the total order, greater than the time stamp and the identifier in the request message, then p sends a reply message to the process that sends the request message. In the opposite case, the process p defers the reply message. Finally, when a process releases the critical section, it must send a reply message to any process with a deferred request.

The Ricart-Agrawala algorithm is derived after four refinements of the abstract model. In the first refinement, a model to request access to the critical section is developed. This model is less nondeterministic that the abstract model, but it is still unfair for processes with the same time stamp. The second refinement introduces a total order among the process in state *waiting*, which allows a fair solution to process with the same time stamp. The first two refinements implement the main ideas of the Ricart-Agrawala algorithm, but all state variables are shared data structures, accessed by all the processes in the system. The goal of the next refinements is to distribute these data structures among the processes, in a way that processes only access local variables, and communication channels are introduced to share information. So, the third refinement introduces the reply messages to grant access to the critical section. Finally, in the fourth refinement, the request messages are introduced, and the Ricart-Agrawala algorithm is derived. The following sections present these refinements.

6.3.1 Requesting Access to the Critical Section

In this refinement, named *Ricart-Agrawala_1*, two issues are considered: the requests produced when processes ask for access to the critical section, and the nondeterminism in the access. In this refinement, there is no request messages and therefore, there is no time stamps associated with request messages. Instead of request messages, a relation is introduced, which models the requests for access to the critical section. If a process p requests access, it is related to all the processes in the system, and a new relation $\{p\} \times \overline{\{p\}}$ is added to the relation. When a process o allows p to enter the critical section, the pair $p \mapsto o$ is removed from the relation. So, a process, waiting for access to the critical section, which is not related any more to other processes, can access the critical section. The time stamp associated with any pair $p \mapsto o$ in the request relation is the time $t(p)$. In this refinement, the updates of logical clocks are constrained to guarantee the access to the critical section in a fair way.

When a process p asks for access to the critical section, the new value assigned to its logical clock depends on the relation among p, and the processes in state *waiting*. If all processes in Wtg are related to p, the new value of its clock must be only greater than its current value. However, if there is a strict subset s of Wtg, such that, all processes in s are not related to p, then, the new value of the logical clock must be greater than the maximum of the time of processes in s. The choice of the new value, allows p to enter the critical section after process in state *waiting*, whose time is lower than the logical clock of p. In other words, any request from a process with a time lower than $t(p)$, is honored before the request of p.

If a process o is related to another one p in the aforementioned relation, p must compare its logical clock with the time of o, to decide if p allows or not, the access of o to the critical section. If p is in state *idle*, or the time of p is greater than or equal to the time of o, the process p must grant access to o. In this case, the couple $o \mapsto p$ must disappear from the relation. However, if p is not in the state *idle*, and its time is lower than or equal to the time of o, then p must defer the access of o. In this case, the request of o must be recorded in a set of deferred requests. This set is used by p, when it leaves the critical section, to remove

any relation among the processes in the deferred set and p.

Two new variables are introduced in this refinement: rq, the relation used to model the requests for access, and df, a function used to record the deferred requests of any process. The type of these variables is as follows:

$$rq \in PR \leftrightarrow PR$$
$$df \in PR \rightarrow \mathbb{P}(PR)$$

In the initial state, rq is assigned the empty relation, and for any process p, $df(p)$ is given the empty set. The following properties, stated as invariants in this refinement, give more information about these variables:

$$rq \cap id(PR) = \varnothing,$$
$$\mathsf{dom}(rq) \subseteq Wtg,$$
$$\forall (p, q) \cdot (p \in Wtg \wedge q \in Wtg - rq^{-1}[\{p\}] \Rightarrow t(q) \leq t(p)),$$
$$\forall p \cdot (p \in Idl \Rightarrow df(p) = \varnothing),$$
$$\forall p \cdot (p \in PR \Rightarrow p \notin df(p))$$

The first invariant indicates that the request relation is not reflexive. The second invariant specifies that processes which are not in state *waiting* are not in the domain of rq. The third invariant gives the order of time for processes in state *waiting*. It states that if processes p and q are in state *waiting*, and q is not related to p ($q \mapsto p \notin rq$), then the time of q is lower than or equal to the time of p. The fourth invariant indicates that, for any process p, if the deferred set of p is not empty, then p is not in state *idle*. The last invariant states that any process p is not in its deferred set.

Events of the refinement *Ricart-Agrawala_1* are presented in figure 6.4. They code the informal behavior of the refinement presented above. The guard of event *req* tests the set $Wtg - rq^{-1}[\{p\}]$; if it is empty ($Wtg \subseteq rq^{-1}[\{p\}]$), the new value of the clock must be only greater than $t(p)$, otherwise, the new value must be greater than the greatest time of processes in the set $Wtg - rq^{-1}[\{p\}]$. When event *req* is executed, the relation $p \times \overline{p}$ is added to the relation rq, which means that process p requests access to all processes in the system. The guard of event *ent* is enabled when there is a process in state *waiting*, which is not in the domain of rq, and the critical section is empty. In this case, the request of that process has been granted by all processes in the system, and it can access the critical section. When a process p leaves the critical section, event *rel* is executed, and any pair $o \mapsto p$ in rq, for any o in the deferred set of p, is removed from rq. It means that p does not block any more, the access of processes in $df(p)$.

Two new events appears in this refinement: *reql* (*request low*) and *reqh* (*request high*). These events implement, in an abstract way, the access mechanism to the critical section. *reql* is enabled when a process p, grants the request made by a process o. This event is enabled when p is in state *idle*, or its time is greater than or equal to the time of o. When *reql* is executed, the pair $o \mapsto p$ is removed from rq. On the other hand, *reqh* is executed when a process p defers the request of a process o. This event is enabled when p is not in state *idle*, and its time is lower than, or equal to the time of o. When this event is executed, process o is added to the deferred set of p. It must be remarked that *Ricart-Agrawala_1* behaves not deterministically when p is not in state *idle*, and the time stamps of p and o are equal. In this case, the request of o can be granted or deferred. This issue is considered in the next refinement.

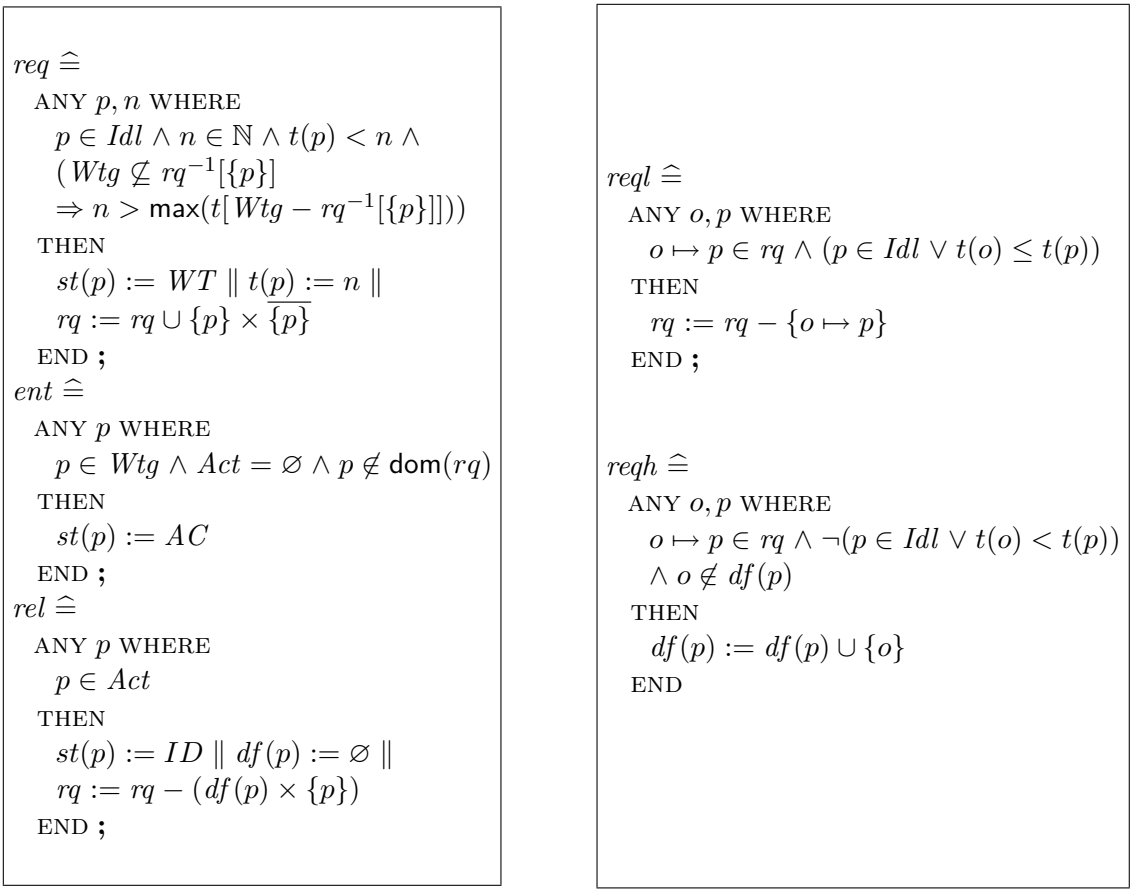

Figure 6.4: System *Ricart-Agrawala_1*

Preserving Liveness

In this section the preservation of the main liveness property (6.1) in the refinement *Ricart-Agrawala_1* is proved. As it has been stated, the proof of this property depends on the preservation of properties (6.6) and (6.7). However, as the guard of event *rel*, which is the helpful event in the property (6.7), does not change in this refinement, the preservation of (6.7) holds trivially, as stated by rule R1. Therefore, only the preservation of (6.6) must be proved.

According to LIP and SAP proof obligations, the following properties are sufficinet to prove the preservation of property (6.6):

$$q \in Wtg \wedge Act = \varnothing \wedge \neg grd(ent_1) \rightsquigarrow grd(ent_1) \tag{6.12}$$

$$q \in Wtg \wedge Act = \varnothing \wedge grd(ent_1) \Rightarrow [e]\, grd(ent_1) \tag{6.13}$$

where e in (6.13) is universally quantified over the set of events $\{req, rel, reql, reqh\}$ and $grd(ent_1)$ denotes the guard of event *ent* in this refinement, and it is equivalent to $Act = \varnothing \wedge Wtg \not\subseteq \mathsf{dom}(rq)$. Informally, this guard indicates that execution of *ent* is only possible if the critical section is empty and at least, one requesting process must have access rights from all the processes in the system.

As (6.13) is easily proved using predicate and substitution calculus, only the proof of (6.12)

is considered here. In fact, (6.12) specifies the main liveness property of the access mechanism in this refinement: a requesting process eventually gets access rights from all the processes when the system arrives into a state where the critical section is empty. This behavior is ensured by two facts: the number of requests in *rq* is decremented by executions of event *reql*, and only the execution of *req* increments the number of requests, but this increment is bounded by the number of process in state *idle*. In this way, the repeated execution of *reql*, eventually leads to an state where a process gets access rights from all the processes in the system. The formalization of this argument requires the definition of a variant in a lexicographic order:

$$W = [\mathsf{card}(Idl), \mathsf{card}(rq)] \tag{6.14}$$

where $\mathsf{card}(Idl)$ is the number of processes in state *idle*, and $\mathsf{card}(rq)$ is the number of couples in *rq*. Using this variant, the following basic liveness property is sufficient to prove (6.12):

$$\begin{array}{l} reql \cdot q \in Wtg \wedge Act = \varnothing \wedge Wtg \subseteq \mathsf{dom}(rq) \wedge W = w \gg \\ \quad (q \in Wtg \wedge Act = \varnothing \wedge Wtg \subseteq \mathsf{dom}(rq) \wedge W \prec w) \vee \\ \quad (Act = \varnothing \wedge Wtg \not\subseteq \mathsf{dom}(rq)) \end{array} \tag{6.15}$$

where w stands for the list $[k, l]$, and k and l are universally quantified over the natural numbers. Property (6.15) specifies that execution of *reql* eventually reaches a state where a waiting process gets access rights from all processes in the system and the critical section is empty. Property (6.15) is proved by the WF0 and WF1 proof obligations, and by application of the basic and induction rules, the following property is derived:

$$q \in Wtg \wedge Act = \varnothing \wedge Wtg \subseteq \mathsf{dom}(rq) \leadsto Act = \varnothing \wedge Wtg \not\subseteq \mathsf{dom}(rq)$$

which is equivalent to (6.12), given the definition of $grd(act_1)$.

Proof Obligations for *Ricart-Agrawala_1*

In this table, we present the number of proof obligations of refinement *Ricart-Agrawala_1*. The row labeled "Total Liveness" indicates the number of assertions generated by WF0 and WF1 rules, required in the proof of the property (6.15) and the SAP proof (6.13). The table is as follows:

	NbObv	NbP0	NbPRi	NbPRa
Lemmas	0	1	0	1
Init	3	8	0	8
req	5	6	5	1
ent	9	4	3	1
rel	3	9	6	3
reql	16	8	3	5
reqh	9	4	3	1
Total Refinement	45	40	20	20
Total Liveness	11	23	18	5
Total	56	63	38	25

6.3.2 A Total Order among the Time Stamps

In this refinement, *Ricart-Agrawala_2*, the main concern is the non determinism generated by the time stamps associated with requests. As explained above, this issue generates an unfair choice for processes with the same time stamp, when they can enter the critical section. In order to resolve the non determinism, a total order among the time stamps is introduced. In this order, each process has associated an unique identifier, which is a natural number. So, the couples made up of the time stamps and identifiers of processes, form a total ordered set. On another hand, to simplify the choice of time stamps when a process requests access to the critical section, a new variable is introduced for each process, which records the greatest time stamp known by that process.

A constant function i, of type $PR \rightarrowtail\!\!\!\!\!\rightarrow 0..\mathsf{card}(PR) - 1$, associates an identifier with each process; therefore, for each process p, $i(p)$ is a natural number, which denotes the identifier of p. For each process, a new variable records the value of the highest time stamp, associated with the requests treated by p. This new variable is denoted by ht and it is of type $PR \rightarrow \mathbb{N}$. For each process p, $ht(p)$ has assigned zero as initial value. $ht(p)$ is updated each time that p compares the time stamp of requests with its highest time stamp; after the comparison, the maximum of these values is stored in $ht(p)$. Therefore, when p requests access to the critical section, its request is stamped with the value $ht(p) + 1$, to ensures that p accesses the critical section after any process, known by p, with early time stamps.

For any process p, the highest time stamp recorded in $ht(p)$ is always equal or greater than its local clock. If any process o has granted access rights to p, then the time stamp of p is equal or lower than the highest time stamp of o. When process p is in the critical section, the highest time stamp of any process o is equal or greater than the time stamp of p. These facts are specified as the following invariants:

$$\forall p \cdot (p \in PR \Rightarrow t(p) \leq ht(p))$$
$$\forall (o, p) \cdot (o \in PR \wedge p \in Wtg - rq^{-1}[\{o\}] \Rightarrow t(p) \leq ht(o))$$
$$\forall (o, p) \cdot (o \in PR \wedge p \in Act \Rightarrow t(p) \leq ht(o))$$

The unfair choice of process with the same time stamp is eliminated with the total order. So, the process which accesses the critical section, compared with any other requesting process, is the least process in the order. On another hand, if a requesting process o grants access rights to another one p, then p is lower than q in the total order. These conditions are given as the following invariants:

$$\forall (o, p) \cdot (o \in PR \wedge o \notin Idl \wedge p \in Act \wedge o \neq p \Rightarrow [t(p), i(p)] \prec [t(o), i(o)])$$
$$\forall (o, p) \cdot (o \in PR \wedge o \notin Idl \wedge p \in Wtg - rq^{-1}[\{o\}] \Rightarrow [t(p), i(p)] \prec [t(o), i(o)])$$

When a request of process p is deferred by process o, then the time stamp of p is equal or lower than the highest time stamp recorded by o. Moreover, the request of o is smaller than the request of p in the total order. Under these constraints of order, any deferred request belongs to processes in state *waiting*. The invariants specifying these properties are:

$$\forall (o, p) \cdot (o \in PR \wedge p \in PR \wedge p \in df(o) \Rightarrow t(p) \leq ht(o))$$
$$\forall (o, p) \cdot (o \in PR \wedge p \in PR \wedge p \in df(o) \Rightarrow [t(o), i(o)] \prec [t(p), i(p)])$$
$$\forall p \cdot (p \in PR \Rightarrow df(p) \subseteq Wtg)$$

Events in the *Ricart-Agrawala_2* refinement are presented in figure 6.5. There is no new events in this refinement, and the state variables of the previous refinement are conserved. The guards of events *req* and *ent*, become simpler than the corresponding guards in the previous refinement. Variable *ht* in event *rq* simplifies the new values of the clocks, which are used as time stamps in the requests, while the total order guarantees that the critical section is empty when a requesting process has received access rights from any other process. On another hand, the modification of events *reql* and *reqh* includes the update of the variable $ht(p)$, for a process p who receives a request from process o. The new value corresponds to the maximum between the highest time stamps of p and the time stamp of o.

req $\widehat{=}$
ANY p WHERE
$p \in Idl$
THEN
$st(p) := WT \parallel t(p) := ht(p) + 1 \parallel$
$ht(p) := ht(p) + 1 \parallel$
$rq := rq \cup \{p\} \times \overline{\{p\}}$
END ;

ent $\widehat{=}$
ANY p WHERE
$p \in Wtg \wedge p \notin \mathsf{dom}(rq)$
THEN
$st(p) := AC$
END ;

rel $\widehat{=}$
ANY p WHERE
$p \in Act$
THEN
$st(p) := ID \parallel df(p) := \varnothing \parallel$
$rq := rq - (df(p) \times \{p\})$
END ;

reql $\widehat{=}$
ANY o, p WHERE
$o \mapsto p \in rq \wedge (p \in Idl \vee$
$[t(o), i(o)] \prec [t(p), i(p)])$
THEN
$rq := rq - \{o \mapsto p\} \parallel$
$ht(p) := \mathsf{max}(\{ht(p), t(o)\})$
END ;

reqh $\widehat{=}$
ANY o, p WHERE
$o \mapsto p \in rq \wedge \neg(p \in Idl \vee$
$[t(o), i(o)] \prec [t(p), i(p)])$
$\wedge\ o \notin df(p)$
THEN
$df(p) := df(p) \cup \{o\} \parallel$
$ht(p) := \mathsf{max}(\{ht(p), t(o)\})$
END

Figure 6.5: System *Ricart-Agrawala_2*

At this time, the access mechanism is completly deterministc: Always the least requesting process in the total order of time stamps accesses the critical section. The following section shows that the main liveness property is preserved in this refinement.

Preserving Liveness

The guard of event *ent* in this refinement is weaker than the guard of *ent* in refinement *Ricart-Agrawala_1*. Therefore, according to rule R2, only the preservation of the basic property (6.15) is required in the preservation of the basic property (6.6) in refinement *Ricart-Agrawala_2*. However, since the guard of event *reql* in this refinement is implied by the left hand side of

property (6.15), proved with the least process in state *waiting* as witness:

$$q \in Wtg \wedge Act = \varnothing \wedge Wtg \subseteq \mathsf{dom}(rq) \Rightarrow grd(reql_2) \tag{6.16}$$

according to rule R1, basic property (6.15) is preserved in this refinement, and so, (6.6) is preserved.

Proof Obligations for *Ricart-Agrawala_2*
In this table, we present the number of proof obligations of refinement *Ricart-Agrawala_2*. The row labeled "Total Liveness" indicates the proof of the implication (6.16). The table is as follows:

	NbObv	NbP0	NbPRi	NbPRa
Lemmas	0	5	2	3
Init	5	10	1	9
req	3	11	7	4
ent	9	8	8	0
rel	6	8	8	0
reql	20	24	21	3
reqh	7	9	8	1
Total Refinement	50	75	55	20
Total Liveness	0	1	1	0
Total	50	76	56	20

6.3.3 Reply Messages to Grant Access Rights

At this time in the development, the main concepts of the Ricart et Agrawala algorithm have been introduced. The main concern of this refinement and the last one, is the distribution of information among the process in the system. In this third refinement, named *Ricart-Agrawala_3*, a channel is introduced which carries the replies of processes when they grant access rights. Moreover, a new variable is introduced for each process, which determines the number of reply messages that a process receives. In this way, any requesting process can access the critical section, when the number of pending requests is zero.

A reply channel is modeled by a subset of processes. Each process has assigned a reply channel. Reply channels are modeled by a new variable *chp* of type $PR \rightarrow \mathbb{P}(PR)$. Initially, the reply channel of any process is empty. A constant function *snp*, of type $PR \rightarrow (\mathbb{P}(PR) \rightarrow \mathbb{P}(PR))$, is used to model the transmission of reply messages. A reply message is simply a value of type PR, which denotes the process which grants access rights. So, for any process p and reply channel ch, which is a subset of PR, $sdp(p)(ch)$ denotes the channel $ch \cup \{p\}$. A reply messages is sent in three circumstances: when a process in state *idle* receives a request, or it is not in state *idle*, but the time stamp of the request is smaller than the local clock, or a process leaves the critical section, and has deferred requests. The reply channel of processes which are not in state *waiting* is empty. On another hand, any reply o, in the reply channel of p, means that p is not requesting access to o any more. These two properties of the request channels are specified as two invariants:

$$\forall p \cdot (p \in PR \wedge p \notin Wtg \Rightarrow chp(p) = \varnothing)$$
$$\forall p \cdot (p \in PR \Rightarrow rq[\{p\}] \subseteq PR - chp(p))$$

The number of pending requests in each process is recorded in a new variable *orc*. It is a variable of type $PR \to \mathbb{N}$. Initially, the pending request count of any process is $\mathsf{card}(PR) - 1$, which means that any process asking for access to the critical section, must wait for $\mathsf{card}(PR) - 1$ reply messages. A new event, named *rpl* (*reply*), is introduced, which updates $orc(p)$ when process p receives a reply message; moreover the execution of *rpl* removes the message from the reply channel, which models the reception of reply messages. Therefore, the guard of event *ent*, executed when a process enters the critical section, is enabled when the pending request count, of any requesting process, is zero. This test is done only with local variables, and it does not access a shared variable, as in the previous refinement. On another hand, in this refinement the deferred set is implemented by a boolean array. So, a new variable *rd*, of type $PR \to (PR \to BOOL)$ is introduced. For any process o and p, $rd(p)(o)$ is *true* if the access request from o is deferred by p. In the initial state, the array of any process is *false*. The gluing invariant which relates *df* with the variable *rd* is:

$$\forall p \cdot (p \in PR \Rightarrow df(p) = rd(p)^{-1}[\{true\}])$$

The following invariants give more information among the new variables introduced:

$$\forall p \cdot (p \in PR \Rightarrow rd(p)^{-1}[\{TRUE\}] \subseteq Wtg \cap orc^{-1}[\mathbb{N}_1])$$
$$\forall p \cdot (p \in Wtg \wedge orc(p) = 0 \Rightarrow chp(p) = \varnothing)$$
$$\forall p \cdot (p \in Wtg \Rightarrow \mathsf{card}(rq[\{p\}]) = orc(p) - \mathsf{card}(chp(p)))$$

The first invariant states that deferred requests belongs to processes in state *waiting* with pending requests. The second one specifies that any process, with a non empty reply channel, has pending requests. Finally, the third invariant relates, for any process in state *waiting*, the number of requests not yet granted with the number of pending requests and the number of reply messages.

Events in the *Ricart-Agrawala_3* refinement are presented in figure 6.6. The transmission of reply messages in events *reql* and *rel* is specified by statements of the form $chp := chp \lhd\!\!+ chp'$, which means that only the reply channel of process in the set chp' are modified. chp' is in fact a function taking the form $(s \lhd chp \ ; \ snp(p))$, where s is a subset of PR. chp' specifies that the reply message p is sent to the reply channel of each process in s. In event *reql*, only the reply channel of o is modified, while in event *rel*, the reply channels of all processes in the deferred set are modified, if it is not empty.

Events *reql* and *reqh*, which implement a part of the access mechanism, read and write the request relation *rq*. It is the only variable which is shared among the process in the system. Events *ent* and *rpl*, which implement other parts of the access mechanism, only require the access to local variables. Therefore, the access mechanism is not yet completely distributed among the processes. However, the next section gives the proof that the access mechanism preserves the main liveness property of the abstract model.

Preserving Liveness

In this refinement, the guard of event *ent* is strengthened, for this reason, the preservation of property (6.6) must be proved. In fact, the guard of event *ent* is $Wtg \not\subseteq orc^{-1}[\mathbb{N}_1]$, indicating that the execution of *ent* is enabled only when a process in state *waiting* does not have pending requests, and under the assumptions of the invariant, it implies the guard of *ent* in the previous refinement. Therefore, according to LIP and SAP proof obligations, the following

$req \mathrel{\widehat{=}}$
ANY p WHERE
$p \in Idl$
THEN
$st(p) := WT \parallel t(p) := ht(p) + 1 \parallel$
$ht(p) := ht(p) + 1 \parallel$
$orc(p) = \mathsf{card}(PR) - 1 \parallel$
$rq := rq \cup \{p\} \times \overline{\{p\}}$
END ;

$ent \mathrel{\widehat{=}}$
ANY p WHERE
$p \in Wtg \wedge orc(p) = 0$
THEN
$st(p) := AC$
END ;

$rel \mathrel{\widehat{=}}$
ANY p WHERE
$p \in Act$
THEN
$st(p) := ID \parallel$
$rq := rq - (rd(p)^{-1}[\{true\}] \times \{p\}) \parallel$
$rd(p) := PR \times \{false\} \parallel$
$chp := chp \lhd\!\!+ (rd(p)^{-1}[\{true\}] \lhd$
chp ; $snp(p))$
END ;

$reql \mathrel{\widehat{=}}$
ANY o, p WHERE
$o \mapsto p \in rq \wedge (p \in Idl \vee$
$[t(o), i(o)] \prec [t(p), i(p)])$
THEN
$rq := rq - \{o \mapsto p\} \parallel$
$ht(p) := \mathsf{max}(\{ht(p), t(o)\}) \parallel$
$chp := chp \lhd\!\!+ (\{o\} \lhd chp$; $snp(p))$
END ;

$reqh \mathrel{\widehat{=}}$
ANY o, p WHERE
$o \mapsto p \in rq \wedge \neg(p \in Idl \vee$
$[t(o), i(o)] \prec [t(p), i(p)])$
$\wedge\ rd(p)(o) = false$
THEN
$rd(p)(o) := true \parallel$
$ht(p) := \mathsf{max}(\{ht(p), t(o)\})$
END ;

$rpl \mathrel{\widehat{=}}$
ANY p WHERE
$p \in PR \wedge chp(p) \neq \varnothing$
THEN
ANY o WHERE
$o \in chp(p)$
THEN
$chp(p) := chp(p) - \{o\} \parallel$
$orc(p) := orc(p) - 1$
END
END

Figure 6.6: System *Ricart-Agrawala_3*

properties are sufficient to prove the preservation of (6.6):

$$q \in Wtg \wedge Act = \varnothing \wedge Wtg \subseteq orc^{-1}[\mathbb{N}_1] \rightsquigarrow Wtg \not\subseteq orc^{-1}[\mathbb{N}_1] \tag{6.17}$$

$$q \in Wtg \wedge Act = \varnothing \wedge Wtg \not\subseteq orc^{-1}[\mathbb{N}_1] \Rightarrow [ev]\ Wtg \not\subseteq orc^{-1}[\mathbb{N}_1] \tag{6.18}$$

where ev in (6.18) is universally quantified over the set of events $\{req, rel, reql, reqh, rpl\}$. As before, (6.18) is easily proved, and only the proof of (6.17) is given.

The main argument to use in the proof of (6.17), is the decrement of the pending request count of processes in state *waiting*, when event *rpl* is executed in any state where $q \in Wtg \wedge Act = \varnothing$ holds. However, in these states, when event *req* is executed, the pending request count of requesting processes is set to the maximum value, and it may appear as an increment to $orc(p)$ for some processes p. In spite of it, the increment is bounded by the number of processes in state *idle* and this quantity, together with the sum of the pending requests count of all the processes in state *waiting*, forms a variant in a lexicographic order. On the other hand, event *rpl* is executed only when its reply channel is not empty. The execution of this event removes a message from this channel, and it may become empty, disabling its guard. For this reason, the proof of (6.17) must consider *reql* as a helpful event, which is able to send messages to the reply channel of requesting processes. In this way, the decrement of the variant guarantees that a requesting process p eventually reaches a state where there is no more pending requests.

Formalizing the previous reasoning requires the definition of the following variant:

$$W' = [\mathsf{card}(Idl), \Sigma p \cdot (p \in Wtg \mid orc(p))] \tag{6.19}$$

where $\mathsf{card}(Idl)$ denotes the number of process in state *idle*, and $\Sigma p \cdot (p \in Wtg \mid orc(p))$ is the sum of pending requests counts of processes in state *waiting*. The following predicates are used in the liveness properties:

$$P \equiv q \in Wtg \wedge Act = \varnothing \tag{6.20}$$

$$Q \equiv Wtg \not\subseteq orc^{-1}[\mathbb{N}_1] \tag{6.21}$$

$$R \equiv \bigcup p \cdot (p \in Wtg \mid chp(p)) = \varnothing \tag{6.22}$$

P denotes the states where q is waiting and there is no processes in the critical section, Q stands for the guard of event *ent*, and R holds when the union of reply channels of requesting processes is empty. The following basic liveness properties hold in this third refinement:

$$reql \cdot P \wedge \neg Q \wedge R \wedge W' = w \gg P \wedge \neg Q \wedge ((\neg R \wedge W' = w) \vee (R \wedge W' \prec w)) \tag{6.23}$$

$$rpl \cdot P \wedge \neg Q \wedge \neg R \wedge W' = w \gg (P \wedge \neg Q \wedge W' \prec w) \vee Q \tag{6.24}$$

where w stands for the list $[k, l]$, and k and l are universally quantified over the natural numbers. (6.23) specifies that the execution of the event *reql*, in a state where q is waiting, the critical section is empty, the guard of *ent* does not hold and the reply channels are empty, ensures that either the variant is unchanged while a waiting process receives a reply message or the variant is decremented. In the case where the variant is unchanged and there exists a reply message in the channel of a waiting process, the property (6.24) ensures that the execution of event *rpl* decrements the variant or event *ent* becomes enabled.

From these basic properties, the proof of (6.17), which guarantees the preservation of (6.6), is as follows:

1. $P \wedge \neg Q \wedge R \wedge W' = w \leadsto P \wedge \neg Q \wedge ((\neg R \wedge W' = w) \vee (R \wedge W' \prec w))$; (6.23) and BRL
2. $P \wedge \neg Q \wedge \neg R \wedge W' = w \leadsto (P \wedge \neg Q \wedge W' \prec w) \vee Q$; (6.24) and BRL
3. $P \wedge \neg Q \wedge R \wedge W' = w \leadsto ((P \wedge \neg Q \wedge W' \prec w) \vee Q) \vee (\mathcal{P} \wedge \neg Q \wedge R \wedge W' \prec w)$; CAN 1, 2
4. $P \wedge \neg Q \wedge R \wedge W' = w \leadsto (P \wedge \neg Q \wedge W' \prec w) \vee Q$; 3, weakening lhs
5. $P \wedge \neg Q \wedge W' = w \leadsto (P \wedge \neg Q \wedge W' \prec w) \vee Q$; DSJ, 4 and 2
6. $P \wedge \neg Q \leadsto Q$; IND 5

Proof Obligations for *Ricart-Agrawala_3*
In this table, we present the number of proof obligations of refinement *Ricart-Agrawala_3*. The row labeled as "Total Liveness" indicates the number of assertions generated by WF0 and WF1 rules, required in the proof of the property (6.23 and (6.24) and the SAP proof (6.18). The table is as follows:

	NbObv	NbP0	NbPRi	NbPRa
Lemmas	0	1	1	0
Init	4	12	0	12
req	7	8	6	2
ent	13	5	5	0
rel	6	10	8	2
reql	26	21	18	3
reqh	13	4	4	0
rpl	10	9	6	3
Total Refinement	50	75	55	20
Total Liveness	11	23	23	0
Total	90	93	71	22

6.3.4 Request Messages Requesting for Access Rights

In this last refinement, named *Ricart-Agrawala_4*, the distribution of the access mechanism is achieved by the introduction of request channels, which replace the request relation of the previous refinements. Moreover, statements in the specification are ordered by sequential composition, and the parallel composition disappears from the specification. Finally, the name of certain variables are changed, to use the names of the original paper where the Ricart and Agrawala algorithm was presented [55].

When processes ask for access to the critical section, they send request messages to all processes. A request message is a couple of values of type $PR \times \mathbb{N}$, which denotes the process who makes the request, and the time stamp associated with it. Therefore, request channels are modeled by a variable *chq* of type $PR \to \mathbb{P}(PR \times \mathbb{N})$. In order to model the transmission of request messages, a constant function *snq*, of type $(PR \times \mathbb{N}) \to (\mathbb{P}(PR \times \mathbb{N}) \to \mathbb{P}(PR \times \mathbb{N}))$, is used. For a request channel *ch*, process p and time stamp n, $snq(p, n)(ch)$ denotes the channel $ch \cup \{p \mapsto n\}$, which models the transmission of the message $p \mapsto n$ over the channel *ch*.

Reply channels replace the request relation from the previous refinements. The gluing invariant relating the variable *chq* with *rq* is as follows:

$$rq = \bigcup p \cdot (p \in PR \mid (\mathsf{dom}(chq(p)) \cup rd(p)^{-1}[\{true\}]) \times \{p\})$$

This invariant states that for any pair of processes $o \mapsto p$ in *rq*, the request from o is either deferred by p, or it is still in the request channel of p. On another hand, the variables t and ht, which denote time stamps, and highest time stamps respectively, are renamed to *osn* ("our sequence number") and *hsn* ("highest sequence number") in this last refinement; this change is specified by the following gluing invariants:

$$t = osn \wedge ht = hsn$$

The following invariants are needed to achieve the correctness proof of this refinement:

$$\forall p \cdot (p \in PR \Rightarrow chq(p) \in PR \nrightarrow \mathbb{N})$$
$$\forall (o, p, n) \cdot (o \in PR \wedge p \in PR \wedge n \in \mathbb{N} \wedge p \neq o \wedge p \mapsto n \in chq(o) \Rightarrow rd(o)(p) = false)$$
$$\forall (o, p, n) \cdot (o \in PR \wedge p \in Wtg \wedge n \in \mathbb{N} \wedge p \neq o \wedge p \mapsto n \in chq(o) \Rightarrow n = osn(p))$$

The first invariant specifies that each reply channel is in fact a partial function. This constraint prevents two or more request messages, from the same process, in a request channel at the same time. The second invariant states that a request cannot be deferred if it is still in the request channel. Finally, the third invariant indicates that the sequence number in a request message sent by p, is indeed the time stamp in $osn(p)$.

Events of the *Ricart-Agrawala_4* refinement are presented in figure 6.7. The transmission of request messages in event *req* is specified in a similar way as the transmission of reply messages, but instead of use the function *snp*, the function *snq* is used. The request message is sent to any process in the system, except the source of the message ($\overline{\{p\}}$ in the specification). Events *reql* and *reqh* are enable when request channels are not empty, and they do not access directly the variable *osn* of the process who sends the request, as in the previous refinement; the sequence number is accessed from the message in the request channel of each process. At this time, the access mechanism is completely distributed among the process of the system, and the concerned events access only local variables.

It must be remarked that guards of events *reql* and *reqh* take the form $A \wedge B$ and $A \wedge \neg B$ and that some statements are common to the two events. Therefore, these events can be merged to form a single event for the reception of request messages, with a if-then-else statement, as it is done in [3]. The figure 6.8 shows the merge of these events in a single event *rec_req*.

The following section gives the proof that the main liveness property is preserved in this last refinement.

Preserving Liveness

In this last refinement, the guard of event *ent* does not change. Therefore, by R2, the preservation of property (6.6) depends on the preservation of properties (6.23) and (6.24). As the guards of events *reql* and *reqh* are the only ones modified in this refinement, by R1, the preservation of property (6.24) holds trivially. However, the preservation of (6.23) must be proved in this refinement.

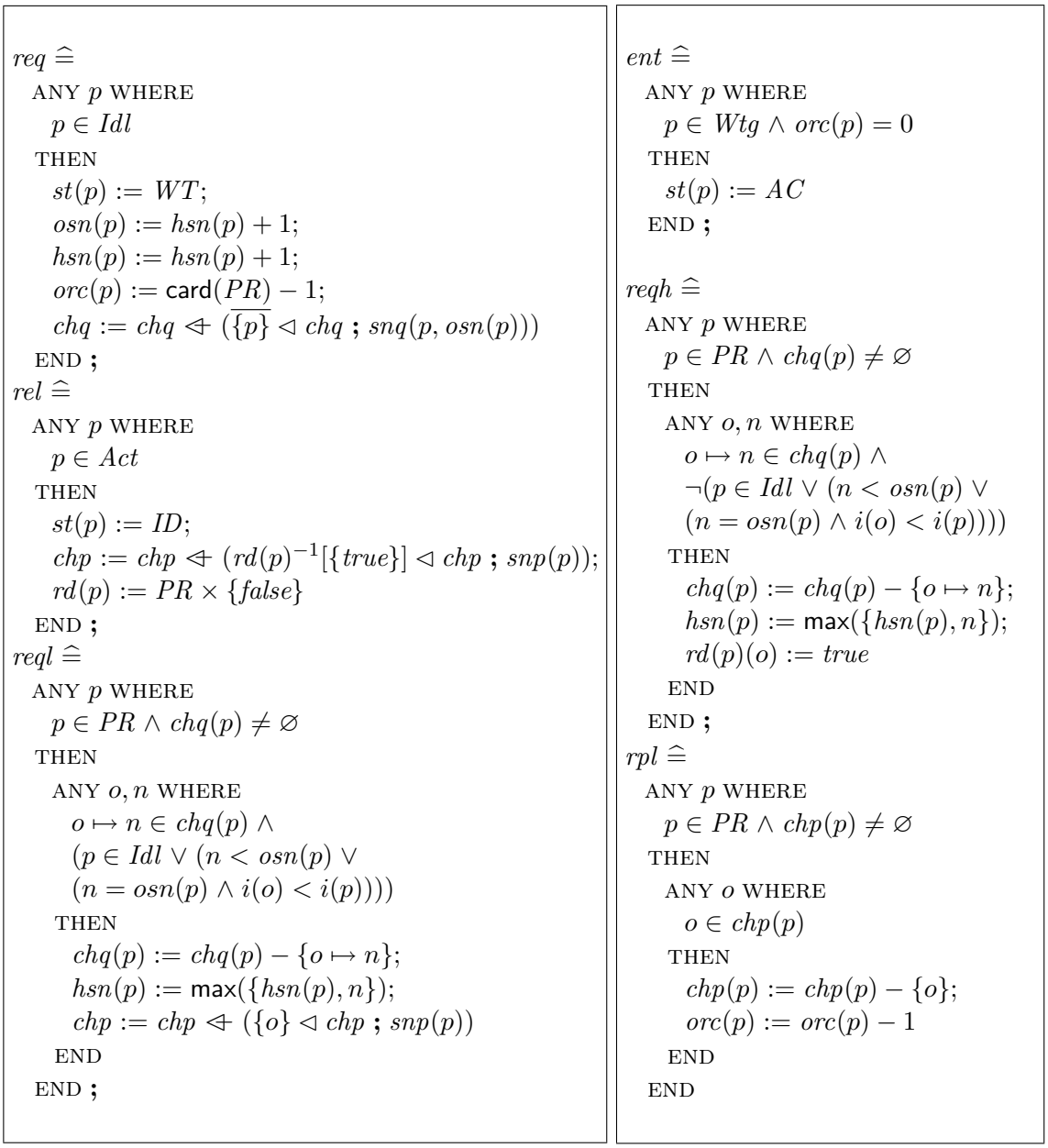

Figure 6.7: System Ricart-Agrawala_4

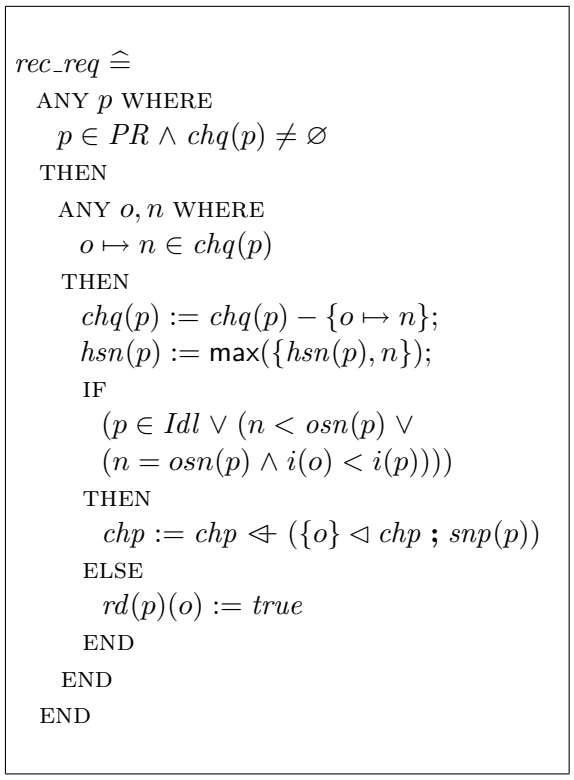

Figure 6.8: Merge of events *reql* and *reqh*

The guard of event *reql* in this refinement is:

$$grd(reql) \equiv \exists(o,p,n) \cdot (p \in PR \wedge o \mapsto n \in chq(p) \wedge (p \in Idl \vee [n, i(o)] \prec [osn(p), i(p)])) \quad (6.25)$$

In fact, it is proved that $grd(reql)$ follows from $P \wedge \neg Q \wedge R$, where the predicates P, Q and R are defined in (6.20), (6.21) and (6.22) respectively. This implication is demonstrated as follows: when all reply channels are empty, and all processes in state *waiting* have pending requests, the least requesting process o, in the total order, cannot have a deferred request. On the other hand, another process p, has at least the request of o in its request channel, because $orc(o) \neq 0$. If p is in state *idle*, the guard of *reql* follows. If p is not in state *idle*, it must be in state *waiting*, but it must be greater than o in the total order of requesting processes, and $grd(reql)$ follows again. Therefore, o can be taken as a witness in the proof of $P \wedge \neg Q \wedge R \Rightarrow grd(reql)$. Now, from this implication, and rule R1, follows the preserevation of the basic property (6.23).

Proof Obligations for *Ricart-Agrawala_4*

In this table, we present the number of proof obligations of refinement *Ricart-Agrawala_4*. The row labeled as "Total Liveness" indicates the number of assertions required to prove the implication $P \wedge \neg Q \wedge R \Rightarrow grd(reql)$, needed in the preservation proof of the property (6.23).

The table is as follows:

	NbObv	NbP0	NbPRi	NbPRa
Lemmas	0	2	2	0
Init	6	8	1	7
req	4	10	5	5
ent	16	2	2	0
rel	12	3	3	0
reql	20	24	9	15
reqh	8	9	4	5
rpl	16	1	1	0
Total Refinement	82	59	27	32
Total Liveness	0	6	6	0
Total	82	65	33	32

6.4 Conclusion

In this chapter, we illustrate the development by refinement of two classical mutual exclusion algorithms. Moreover, apart from safety properties stated by invariants, we prove liveness properties under weak fairness assumptions, using the rules presented in the chapter 3. The abstract model and its refinements have been completely verified with the "Click'n Prove" prover [4]. Moreover, the WF0 and WF1 proof obligations for basic liveness properties, and the SAP rules for verifying the preservation of basic liveness properties, have been automatically generated, and proved as assertions in the model or refinements.

We start by specifying a highly nondeterministic abstract model satisfying two main properties:

- Safety: there is at most one process in the critical section.
- Liveness: any process requesting access to the critical section eventually gets it.

From the abstract model, we show two streams of refinements. In the first one, we derive a centralized algorithm implementing a first input first output policy in the access to the critical section. In the second one, we derive the distributed mutual exclusion algorithm of Ricart-Agrawala.

The abstraction *Mutex*, considers a system with three events modeling requests, accesses and releases of the critical section. The processes use a time stamp to order their access requests. The requesting process with the lowest time stamp gets access to the critical section.

The first flow of refinements is made up of two refinements. The first refinement, *Fifo_1*, introduces an infinite sequence which replaces the time stamps of processes. In this way, processes enter the critical section in a first input first output basis. The second refinement, *Fifo_2*, introduces a fourth event to model a server and the infinite sequence is replace by a finite one. Waiting processes are queued in the sequence. The server allocates a token to the first process in the sequence; it grants access to the critical section. When a process leaves the critical section, it releases the token. The following table gives the number of proof obligations for safety ("Total Safety") and liveness ("Total Liveness") properties generated

in this stream:

	NbObv	NbP0	NbPRi	NbPRa
Total Safety	48	67	34	33
Total Liveness	2	11	6	5
Total	50	78	40	38

The second stream is made up four refinements. The first refinement, *Ricart-Agrawala_1*, introduces a relation to model the broadcast of messages requesting access for the critical action. A process enters into the critical section if it is empty and if all processes have acknowledged its request. Two new events are introduced in this refinement for treating the access requests. The second refinement, *Ricart-Agrawala_2*, introduces a total ordering among the time stamps. This ordering removes the nondeterminism of the previous refinement and allows the access to the critical section to the process with the lowest time stamp. In this refinement, processes still verify the relation modeling the broadcast of request messages before entering the critical section. The third refinement, *Ricart-Agrawala_3*, introduces a new event to model the transmission of reply messages to grant access rights to processes waiting for the critical section. In this way, when a process receives a reply message from all processes, it can access the critical section. Finally, in the last refinement, *Ricart-Agrawala_4*, the relation modeling the broadcast of request messages is removed from the refinement, and an explicit transmission of request messages is introduced. In this refinement, memory is localized to each process. Apart from data structures modeling communication channels, there is no shared data structures and the algorithm can be implemented as network of communicating processes. The following table gives the number of proof obligations for safety and liveness properties generated in this stream:

	NbObv	NbP0	NbPRi	NbPRa
Total Safety	256	244	150	94
Total Liveness	22	53	48	5
Total	278	297	198	99

From the two previous tables, and the table for the *Mutex* model, we obtain the total number of proofs in this development:

	NbObv	NbP0	NbPRi	NbPRa
Total Safety	311	327	184	143
Total Liveness	26	73	61	12
Total	337	400	245	155

Chapter 7

Conclusions and Future Work

7.1 Version française

7.1.1 Résumé

Nous proposons une approche pour la spécification et développement de systèmes d'actions fondée sur le formalisme du B *événementiel. L'approche incorpore une logique temporelle, dans le style* UNITY*, pour spécifier et vérifier des propriétés de vivacité dans un modèle abstrait et ses raffinements, sous les hypothèses de progrès minimal et d'équité faible. La sémantique des opérateurs logiques est définie en termes d'itérations de transformateurs d'état. La vérification des propriétés de vivacité joue un rôle fondamental dans notre approche. Nous proposons des règles de preuve pour vérifier des propriétés de vivacité et pour les préserver dans les raffinements, et nous prouvons la cohérence de ces règles. Afin d'atteindre nos objectifs, dans cette thèse, nous procédons de la manière suivante.*

Dans le chapitre 3, nous commençons par la proposition de règles de preuve pour vérifier des propriétés de vivacité de base sous les hypothèses de progrès minimal (MP0 et MP1) et d'équité faible (WF0 et WF1). Ces règles sont fondées sur le calcul des plus faibles préconditions et elles peuvent être vérifiées par le prouveur Click'n Prove. Des propriétés de vivacité générales sont alors dérivées à partir des propriétés de base par l'application des règles issues de la logique de programmation UNITY*. L'approche de la spécification et preuve des propriétés de vivacité est illustrée par une série d'exemples. Afin de garantir que les propriétés de vivacité, prouvées dans l'abstraction, continuent à être vraies dans un raffinement, les règles BMP et LMP sont proposées pour préserver les propriétés de vivacité de base sous la condition de progrès minimal, et les règles LIP et SAP, pour préserver ces propriétés sous la condition d'équité faible. L'application de ces règles est illustrée dans une autre série d'exemples. Tous ces exemples sont vérifiés avec le prouveur Click'n Prove. Les obligations de preuve pour la vérification des propriétés de vivacité de base sont automatiquement générées par une boite à outils, et elles sont placées en tant qu'assertions dans des machines* B*, afin d'être vérifiées par le prouveur. Dans ce chapitre nous faisons l'hypothèse que la préservation des propriétés de vivacité de base est une condition suffisante pour préserver des propriétés de vivacité générales. Cette hypothèse est prouvée dans l'annexe D.*

Dans le chapitre 4, nous donnons des arguments formels de notre approche. Le traitement est présentée dans un cadre ensembliste théorique qui utilise des transformateurs d'ensembles pour dénoter la plus faible précondition et la plus faible précondition libérale des événements.

Un transformateur d'état non classique, nommée l'opérateur dovetail *est introduit pour modéliser un choix équitable sous l'hypothèse d'équité faible. En utilisant ces notions, nous introduisons la sémantique de nos formules temporelles en termes de points fixes de transformateurs d'ensembles qui dénotent des itérations d'événements. Ces transformateurs d'ensembles permettent la définition d'une relation de* terminaison *qui est utilisée pour interpréter des propriétés* leads-to*. D'autre part, les propriétés de vivacité satisfaites par un système sont dénotées par une autre relation entre sous-ensembles d'états nommée relation d'*atteignabilité*. Afin de prouver la cohérence des règles pour* leads-to*, nous prouvons que la relation d'*atteignabilité *est contenue dans la relation de* terminaison*. D'autre part, la preuve de la complétude relative est donnée par la démonstration de l'inclusion dans le sens contraire. Dans un premier temps, ces preuves ne considèrent pas des hypothèses d'équité. Le théorème 2, qui énonce la cohérence et la complétude des règles* leads-to*, énonce uniquement les prémisses qui doivent être satisfaites par les transformateurs d'ensembles qui modèlent l'itération d'événements. Ce théorème est alors instancié pour considérer des hypothèses d'équité. Finalement, dans ce chapitre nous prouvons que les règles utilisées pour vérifier les propriétés de vivacité de base sont cohérentes dans notre cadre. Dans l'annexe B, nous énonçons à nouveau notre sémantique afin de considérer le plus fort invariante [62]. Dans l'annexe C nous montrons certaines preuves qui ne sont pas présentées dans le chapitre 4. Nous remarquons le fait que quelques preuves ont été mécaniquement prouvées par le prouveur Click'n Prove. En effet, notre cadre ensembliste théorique peut être codé directement en tant que formules dans la notation* B*, et alors nous pouvons prouver des théorèmes dans une machine abstraite. Nous utilisons la clause* CONSTANTS *pour déclarer des noms et la clause* PROPERTIES *pour déclarer des informations de typage et quelques axiomes. Alors, nous énonçons nos lemmes dans la clause* ASSERTIONS *et nous procédons à la preuve avec le prouveur. L'annexe F montre les machines* B *d'une de ces preuves.*

Le principal résultat du chapitre 4 est la preuve que les notions d'atteignabilité et de terminaison sont des notions équivalentes. Cela signifie que nous pouvons prouver une propriété $p \mapsto q \in \mathfrak{L}_{\mathcal{E}}$ par la preuve de l'inclusion de p dans l'ensemble de terminaison de l'itération d'événements qui termine dans q sous une certaine condition d'équité et vice versa. Pour cette raison, afin de prouver qu'une propriété leads-to *$p \mapsto q \in \mathfrak{L}_{\mathcal{E}}$ est préservée dans un raffinement, nous pouvons prendre l'approche dans [7], où les auteurs prouvent que $r^{-1}[p]$ est inclus dans l'ensemble de terminaison de l'itération d'événements dans le raffinement qui terminent dans $r^{-1}[q]$. Cependant, comme nous pouvons le constater, la preuve est longue et difficile. En considérant nos résultats, une preuve plus courte peut être obtenue par la considération de la définition inductive de la relation d'atteignabilité $\mathfrak{L}_{\mathcal{E}}$. Dans le chapitre 5, nous prouvons par induction structurelle que BMP and LMP sont des conditions suffisantes pour garantir la préservation des propriétés de vivacité sous la condition de progrès minimal et que LIP et SAP permettent la préservation de la vivacité sous l'hypothèse d'équité faible. Les annexes D et E présentent les preuves concernant la préservation des propriétés de vivacité.*

Finalement, dans le chapitre 6, nous développons un exemple plus important d'un algorithme d'exclusion mutuelle sous l'hypothèse d'équité faible. Le premier objectif de cet exemple est de montrer comment nous appliquons les règles de preuve pour la vérification des propriétés de vivacité dans la spécification et les raffinements. Le deuxième objectif est d'expliquer, par des exemples, comment nous concevons le développement des algorithmes distribués. En effet, notre modèle permet un haut niveau de non déterminisme ; les propriétés de vivacité sont prouvées dans ce modèle abstrait. Par une restriction du non déterminisme, nous proposons deux classes de raffinements. La première classe est appropriée pour une

mise en œuvre du modèle dans un environnement centralisé : un serveur donne accès dans un ordre fifo. *La seconde classe de raffinements est adaptée à une mise en œuvre dans un environnement distribué. Nous développons l'algorithme proposé par Ricart et Agrawala, où il n'y a pas un serveur central et où tous les processus dans le système coopèrent pour donner les droits d'accès. Dans ces raffinements, nous prouvons, par les règles proposées, que les propriétés de vivacité dans l'abstraction sont préservées dans les raffinements. Ces exemples sont complètement prouvés avec le prouveur Click'n Prove.*

7.1.2 Positionnement de notre travail

Notre travail est placé dans le contexte du développement de programmes par des pas de raffinement en utilisant le formalisme du B *événementiel. L'approche originale des modalités dans [7] est très restreint. La spécification des modalités a besoin de la définition d'invariants et de fonctions variantes. La raison de ces restrictions est que le modèle de calcul, utilisé pour raisonner sur les modalités, est une construction du type* do od *sans hypothèses d'équité. Pour cette raison, la preuve des modalités devient une preuve de correction totale de cette construction.*

Notre travail améliore l'approche originale des modalités du B *événementiel proposée dans [7], par l'utilisation d'une logique similaire à celle d'*UNITY*, dans la spécification et preuve des propriétés de vivacité, sous les hypothèses de progrès minimal et d'équité faible. Comme cela est illustré par des exemples du chapitre 3, nous avons plus de flexibilité dans la spécification et la preuve des propriétés de vivacité. L'approche de spécification et de preuve des propriétés de vivacité dans les systèmes* B *événementiels présenté dans cette thèse, permet des preuves modulaires où des analyses par cas sont directement supportés par la logique et il n'est pas nécessaire de développer un système par la classe de raffinements comme proposés dans [34].*

Le modèle de calcul dans notre cadre ne correspond pas au modèle de UNITY*, car le gardes des événements ne sont pas toujours habilitées. Pour cette raison, les résultats de la logique* UNITY *dans [21, 22, 39, 46, 36], ne s'appliquent pas directement dans notre cadre si la définition de la relation de base* ensures *ne change pas. Notre définition de la relation* ensures*, dans un modèle plus général où les gardes des actions sont des prédicats, est inspirée de [37]. Cette référence nous a aussi suggéré l'idée de dénoter les propriétés* leads-to *par des transformateurs d'ensembles.*

Nos transformateurs d'ensembles pour les hypothèses de progrès minimal ou équité faible sont définies d'une manière originale. Nos définitions approfondissent et clarifient l'idée de [7] et [12], à propos de la relation entre terminaison d'itération d'événements et d'atteignabilité, comme celle reflétée par des propriétés leads-to*. Mis à part les transformateurs d'états qui dénotent les actions d'un système, nos transformateurs d'états pour* leads-to *sont donnés d'une manière constructive et utilisent des transformateurs d'états primitifs tels que l'itération, le choix, le choix équitable, la séquence, la garde et la précondition, et ne sont pas uniquement une équation de points fixes, comme les transformateurs de prédicats qui ont été décrit dans les paragraphes précédents. Nous prouvons que nos définitions satisfont les exigences d'un système cohérent et relativement complet pour les propriétés* leads-to*, comme cela a été faite dans [37].*

Le choix des définitions des propriétés leads-to *sous les hypothèses de progrès minimal ou d'équité faible, est le résultat de notre approche au développement de programmes par pas de raffinements. En effet, nous espérons que des systèmes sous des conditions d'équité faible seront mis en œuvre par des raffinements où les événements itèrent sous l'hypothèse de*

progrès minimal. Pour cette raison, nous avons été intéressé dans la définition de transformateurs d'ensembles en relation avec l'itération d'événements. De cette manière, le raffinement d'événements nous donne des renseignements sur les conditions suffisants pour garantir le raffinement des propriétés de vivacité [59]. En outre, lorsque une propriété leads-to *est satisfaite sous la condition d'équité faible, et que le raffinement contient quelques mécanismes pour assurer un entrelacement équitable d'actions, nous garantissons que, par des conditions imposées aux événements dans le raffinement, les propriétés* leads-to *sont encore satisfaites sous la condition de progrès minimal. Dans cette thèse nous avons énoncé formellement ces faits par un règle de preuve qui place les idées de transformation de programmes pour la mise en œuvre d'équité faible, présentées dans [8], dans le cadre du raffinement.*

7.1.3 Travail futur

Les propriétés de vivacité sous l'hypothèse d'équité forte manque manifestement dans notre travail. En suivant notre approche, nous avons besoin d'un opérateur de choix similaire à l'opérateur dovetail*, qui permet un choix fortement équitable d'événements utiles. Il doit permettre la définition d'une relation de* terminaison *et de cette manière nous devions définir la relation de base avec cette hypothèse d'équité. Nous avons besoin de chercher la plus faible précondition libérale et l'ensemble de terminaison de cet opérateur afin de déterminer le transformateur d'état associé à sa plus faible précondition.*

En permettant des hypothèses d'équité différentes dans le développement des systèmes B *événementiels, nous donnons la possibilité de faire des raffinements où les hypothèses d'équité changent entre deux pas de raffinement. C'est-à-dire, une certaine hypothèse d'équité peut être mise en œuvre dans un raffinement par un ordonnanceur particulier, comme cela est fait dans [8, 9]. Dans ce cas, nous avons besoin de quelques règles pour garantir que les propriétés de vivacité prouvées sous une certaine hypothèse d'équité, sont préservées lorsqu'on change ces hypothèses dans un autre raffinement. Dans la section 4.4.3 nous proposons une règle qui affirme qu'une propriété $P \leadsto Q$, prouvée sous l'hypothèse d'équité faible, est toujours satisfaite par le système lorsqu'on la change par une hypothèse de progrès minimal, et tous les événements du système diminuent un variant dans un état où $\neg Q$ est vrai. Malgré le fait que la cohérence de cette règle est prouvée, il y a des cas où il n'est pas possible de trouver un variant pour satisfaire les conditions de cette règle. Par exemple, dans le cas de la propriété $q \in Wtg \leadsto q \in Act$, spécifiée dans les exemples du chapitre 3, nous ne pouvons pas trouver un variant qui décroît dans le système raffiné dans un état où $q \notin Act$. Nous avons besoin de chercher des règles plus pratiques afin de garantir la mise en œuvre facile d'ordonnanceurs.*

Les exemples traités dans cette thèse, et beaucoup d'autres sont des exemples de systèmes ouvertes : collection de logiciels communicants, matériel et des composants humains. Dans ces exemples, les actions de ces composants son modélisés par des événements atomiques. Le raffinement de ces systèmes incorpore des nouveaux événements et modifie l'espace d'états, mais le système continue à être modélisé d'une manière monolithique : une collection d'événements partageant des variables d'état. Afin d'assurer une mise en œuvre correcte de ces systèmes dans le monde réel, nous avons besoin de faire des raffinements modulaires qui préservent les propriétés de sûreté et de vivacité. Cela est un domaine de recherche ouvert dans lequel nous sommes intéressés.

7.2 English Version

7.2.1 Summary of the Thesis

We propose an approach to the specification and development of action systems founded on the event B formalism. The approach incorporates a temporal logic, in a UNITY-like style, to specify and verify liveness properties in abstract models and their refinements under minimal progress and weak fairness assumptions. The semantics of the logical operators is defined in terms of iterations of set transformers. The verification of liveness properties plays a fundamental role in our approach. We propose proof rules to verify both the basic liveness properties and their preservation under refinement, and we prove soundness of these rules. In order to reach our goals, in this thesis, we proceed as follows.

In Chapter 3, we start with the proposal of proof rules to verify basic liveness properties under minimal progress (MP0 and MP1) and weak fairness (WF0 and WF1) assumptions. These rules are founded on the weakest precondition calculus and can be verified by a tool like Click'n Prove prover. General liveness properties are then derived from basic properties by applying rules issued from the logic programming of UNITY. The approach of specification and verification of liveness properties is illustrated by a series of examples. So as to guarantee that liveness properties proved in the abstraction continue to hold in a refinement, the BMP and LMP rules are proposed to preserve basic liveness properties under a minimal progress condition and the LIP and SAP rules to preserve basic properties under a weak fairness assumption. The applications of these rules are illustrated in another series of examples. All examples are verified with the Click'n Prove prover. The proof obligations for the verification of basic liveness properties are automatically generated by a toolbox and they are placed as assertions in B machines in order to be verified by the prover. In this chapter we claim that preserving basic liveness properties is a sufficient condition to preserve general liveness properties. This claim is proved in appendix D.

In Chapter 4 we give formal arguments about our approach. The treatment is presented in a set theoretical framework using set transformers to denote weakest preconditions and weakest liberal preconditions of events. A non classical set transformer, named the dovetail operator, is introduced to model a fair choice of events under a weak fairness assumption. Using these notions, we introduce the semantics of our temporal formulas in terms of fixpoints of set transformers denoting iteration of events. These set transformers allow the definition of a *termination* relation which is used to interpret *leads-to* properties. On another hand, liveness properties holding in a system are denoted by another relation on subsets of states named *reachability*. This relation is inductively defined and we prove in appendix A that this relation is equivalent to the *leads-to* relation. In order to prove soundness of the *leads-to* rules, we prove that the *reachability* relation is contained in the *termination* relation. On another hand, the relative completeness proof is given by demonstrating the inclusion in the converse direction. In a first time, these proofs do not consider fairness assumptions. Theorem 2, asserting soundness and completeness of the rules for *leads-to*, only states the premises which must be satisfied by set transformer modeling the iteration of events. This theorem is then instantiated to consider fairness assumptions. Finally, in this chapter we prove that the rules used to verify basic liveness properties are sound in our framework. In Appendix B, we restate our semantics by considering the strongest invariant [62]. Appendix C shows certain proofs that are not presented in Chapter 4. We stress the fact that some proofs were mechanically verified with the Click'n Prove prover. In fact, our set theoretical framework can be encoded

directly as formulas in the B notation and then theorems can be proved in a B abstract machine. We use the clause CONSTANTS to declare names and the clause PROPERTIES to give type information and declare some axioms. Then we state our lemmas in the clause ASSERTIONS and we proceed to prove them with the prover. In Appendix F, we show the B machines of some proofs.

The main result of Chapter 4 is the proof that reachability and termination are equivalent notions. It means that we can prove a liveness property $p \mapsto q \in \mathcal{L}_{\mathcal{E}}$ by proving that p is included in the termination set of the iteration of events terminating in q under a certain fairness assumption and vice versa. Therefore, in order to prove that a *leads-to* property $p \mapsto q \in \mathcal{L}_{\mathcal{E}}$ is preserved in a refinement, we can take the approach in [7], where the authors prove that $r^{-1}[p]$ is included in the termination set of the iteration of refined and new events terminating in $r^{-1}[q]$. However, as we can see, the proof is long and difficult. Considering our results, a shorter proof can be achieved by considering the inductive definition of the reachability relation $\mathcal{L}_{\mathcal{E}}$. In Chapter 5 we prove, by structural induction, that BMP and LMP are sufficient conditions to guarantee the preservation of liveness properties under a minimal progress assumption and that LIP and SAP allow the preservation of liveness under a weak fairness assumption. The appendices D and E present the proofs concerning the preservation of liveness properties.

Finally in Chapter 6, we develop a larger example of mutual exclusion algorithms, under a weak fairness assumption. The first goal of this example is to show how we apply the proof rules for verifying liveness properties in the specification and refinement. The second goal is to explain, by examples, how we understand the development of distributed algorithms. In fact, our initial model allows a high level of nondeterminism; safety and liveness properties are proved in this abstract model. By constraining the nondeterminism, we propose two kinds of refinements. The first kind is suitable for an implementation of the model in an centralized environment: a server which grants access in a first input first output basis. The second kind of refinement is adapted to an implementation in a distributed environment. We develop the algorithm proposed by Ricart and Agrawala, where there is not a central server and all processes in the system are cooperating to grant access rights. In these refinements, we prove by the proposed rules, that the liveness property in the abstraction is preserved in the refinement. These examples are completely proved with the Click'n Prove prover.

7.2.2 Positioning of Our Work

Our work is placed in the context of program development by stepwise refinement using the event B formalism. The original approach of modalities in [7] is very restricted. Specification of modalities requires definition of invariant predicates and variant functions. The reason of this restrictions is that the computational model, used to reason about modalities, is a do od construction without fairness assumptions. So, the proof of modalities becomes a proof of total correctness of a do od construction.

Our work improves the original approach of modalities in event B proposed in [7], by using a UNITY-like logic in the specification and proof of liveness properties, under weak fairness or minimal progress assumptions. As illustrated in examples of Chapter 3, it gives more flexibility to the specification and proof of liveness properties. The approach to the specification and proof of liveness in B events systems presented in this thesis, allows modular proofs where cases analysis are directly supported by the logic and there is no need to develop a system by the kind of refinements proposed in [34].

The computational model in our framework does not correspond to the model of UNITY, because the guards of events are not always enabled. Therefore, results of UNITY logic in [21, 22, 39, 46, 36], do not apply directly to our framework if the definition of the basic relation *ensures* is not changed. Our definition of *ensures* relation, in a more general computational model where the guards of actions are state predicates, is inspired from [37]. From this work, we get as well the idea of set transformers to denote *leads-to* properties.

Our set transformers for minimal progress (unfair) or weak fairness assumptions are defined in an original way. Our definition deepens and clarifies the idea of [7] and [12], about the relation between (unfair or weak fair) termination of iteration of events and reachability, as reflected by *leads-to* properties. Apart from set transformers denoting the actions of the system, our set transformers for *leads-to* are constructively built and use primitive set transformers such as iteration, unfair choice, fair choice, sequencing, guard and precondition, and it is not only a fixpoint equation, as the predicate transformers described above. We prove that our definitions fulfill the requirement of a sound and relative complete proof system for *leads-to* properties, as it was done in [37].

The choice of definitions of *leads-to* properties, under minimal progress and weak fairness assumptions, is a result of our approach to program development by stepwise refinement. In fact, we expect that systems under weak fairness assumptions would be implemented, by refinements into concrete systems, where events are iterated under minimal progress assumptions. For this reason, we were interested in defining set transformers for *leads-to* properties in close relation with iteration of events. In this way, refining events gives insight in the sufficient conditions to guarantee the refinement of liveness properties [59]. Moreover, when a *leads-to* property holds in a refinement under weak fairness assumptions, and the refinement contains some mechanisms to ensure a fair interleaving of actions, we can guarantee, by conditions imposed to the events in the refinement, that the *leads-to* property yet holds under minimal progress assumptions. In this thesis we stated formally these facts by a proof rule which replace the ideas of program transformation to implement weak fairness, presented in [8], in the framework of refinement.

7.2.3 Future Work

Liveness properties under strong fairness assumption lack obviously in our work. Following our approach, we need a choice operator, similar to the dovetail operator, allowing a strong fairness choice of helpful events. It would allow the definition of a *termination* relation and then we would define the basic relation under this fairness assumption. We need investigate the weakest liberal precondition and the termination set in order to determine the set transformer associate with its weakest precondition.

Allowing different fairness assumptions in the development of event B systems increases the probability of doing refinements where fairness assumptions change between two steps of refinement. In other words, a certain fairness assumption would be implemented in a refinement by a particular scheduler, as stated in [8, 9]. In this case we need some rules to guarantee that liveness properties proved under a fairness assumption are preserved in the refinement when the fairness assumption change. In section 4.4.3 we propose a rule asserting that a property $P \rightsquigarrow Q$, proved under a weak fairness assumption, holds under a minimal progress assumption if all events in the system decrement a variant in a state where $\neg Q$ holds. In spite that soundness of this rule is proved, there exists cases where it is not possible to find a variant satisfying the conditions of the rule. For example, in the case of the property

$q \in Wtg \rightsquigarrow q \in Act$, specified in the examples of the chapter 3, we cannot find a variant which is decremented in the refined systems in a state where $q \notin Act$. We need to investigate practical rules to guarantee the correct implementation of schedulers.

The examples treated in this thesis an many others are examples corresponding to open systems: collection of interacting software, hardware, and human components. In these examples the actions of these components are modeled by atomic events. The refinement of these systems adds new events and modifies the state space, but the system continues to be modeled in a monolithic way: a collection of events sharing state variables. In order to ensures a correct implementation of these systems in a real world, we need to make modular refinements preserving safety and liveness properties. This is an open subject of research where we are interested.

Bibliography

[1] J.-R. Abrial. Extending B Without Changing it (for Developing Distributed Systems). In *First B Conference*, pages 169–190. Nantes, november 1996.

[2] J.-R. Abrial. *The B-Book, Assigning Programs to Meanings*. Cambridge University Press, 1996.

[3] J.-R. Abrial. Event Based Sequential Program Development: Application to Constructing a Pointer Program. In *FME 2003: Formal Methods, International Symposium of Formal Methods, LNCS 2335*, pages 51–74. Springer-Verlag, September 2003.

[4] J.-R. Abrial and D. Cansell. Click'n Prove: Interactive Proofs within Set Theory. In *Theorem Proving in Higher Order Logics, 16th International Conference, TPHOLs 2003, LNCS 2758*, pages 1–24. Springer-Verlag, 2003.

[5] J.-R. Abrial, D. Cansell, and D. Méry. A mechanically proved and incremental development of IEEE 1394 Tree Identify Protocol. *Formal Aspects of Computing*, (Accepted for publication):, 1992.

[6] J.-R. Abrial, D. Cansell, and D. Méry. Formal Derivation of Spanning Trees Algorithmes. In *ZB 2003: Formal Specification and Development in Z an B, LNCS 2651*, pages 457–476. Springer-Verlag, June 2003.

[7] J.-R. Abrial and L. Mussat. Introducing Dynamic Constraints in B. In *B'98: Recent Advances in the Development and Use of the B Method, LNCS 1393*, pages 83–128. Springer-Verlag, april 1998.

[8] K. R. Apt and E.-R. Olderog. Fairness in Parallel Programs, the Transformational Approach. *ACM Transactions on Programming Languages and Systems*, 10(3):420–455, 1988.

[9] K. R. Apt and E.-R. Olderog. *Verification of Sequential and Concurrent Programs*. Graduate texts in computer science. Springer-Verlag, second edition edition, 1997.

[10] R.-J. Back and R. Kurki-Suonio. Decentralization of Process Nets with Centralized Control. In *2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, pages 131–143, 1983.

[11] R.-J. Back and J. von Wright. *Refinement Calculus A Systematic Introduction*. Graduate Texts in Computer Science. Springer, 1998.

[12] R.-J. Back and Q. Xu. Refinement of Fair Action Systems. *Acta Informatica*, 35:131–165, 1998.

[13] Ralph-Johan Back. Refinement calculus ii: Parallel and reactive programs. In J. W. de-Bakker, W. P. deRoever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems*, volume 430 of *Lecture Notes in Computer Science*, pages 67–93, Mook, The Netherlands, May 1989. Springer-Verlag.

[14] Ralph-Johan Back and Kaisa Sere. Superposition refinement of reactive systems. *Formal Aspects of Computing*, 3:1–23, 1995.

[15] Ralph-Johan Back and Joakim von Wright. Trace refinement of action systems. In B Jonsson and J. Parrow, editors, *CONCUR-94:Concurrency Theory*, volume 836 of *Lecture Notes in Computer Science*, pages 367–384, Uppsala, Sweden, Aug 1994. Springer-Verlag.

[16] D. Bert and H. Ruíz Barradas. Propriétés de vivacité dans les systèmes B. Application à l'algorithme de Ricart-Agrawala. *To appear in Technique et Science Informatique,RSTI,série TSI.*

[17] D. Bert and H. Ruíz Barradas. Développement et preuve de vivacité de l'algorithme distribué de Ricart-Agrawala. In *Actes de la Conférence AFADL'06: Approches Formelles dans l'Assistance au Développement de Logiciels*, pages 161–178. ENST, Paris, France, 2006.

[18] M. Broy and G. Nelson. Adding Fair Choice to Dijkstra's Calculus. *ACM Transactions on Programming Languages and Systems*, 16(3):924–938, May 1994.

[19] M. Butler and M. Waldén. Distributed System Development in B. In *First B Conference*, pages 155–168. Nantes, november 1996.

[20] D. Cansell, G. Gopalakrishnan, M. Jones, D. Méry, and A. Weinzoepflen. Incremental Proof of the Producer/Consummer Property for the PCI Protocol. In *ZB 2002: Formal Specification and Development in Z an B, LNCS 2272*, pages 22–41. Springer-Verlag, January 2002.

[21] K. M. Chandy and J. Misra. *Parallel Program Design. A Foundation.* Addison-Wesley, 1988.

[22] K. M. Chandy and B. A. Sanders. Predicate Transformers for Reasoning about Concurrent Computation. *Science of Computer Programming*, 24:129–148, 1995.

[23] E. W. Dijkstra. Hierarchical Ordering of Sequential Processes. *Acta Informatica*, 1:115–138, 1971.

[24] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[25] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics.* Springer-Verlag, 1990.

[26] S. Dune. Introducing Backward Refinement into B. In *ZB 2003: Formal Specification and Development in Z an B, LNCS 2651*, pages 178–196. Springer-Verlag, June 2003.

[27] S. Chouali F. Bellegarde and J. Julliand. Verification of Dynamic Constraints for B Event Systems under Fairness Assumptions. In *ZB 2002 International Conference, LNCS 2272*, pages 481–500. Springer-Verlag, january 2002.

[28] Nissim Franncez. *Fairness*. Graduate texts in computer science. Springer-Verlag, 1986.

[29] R. Goldblatt. *Logics of Time And Computation, Second Edition*. Center for the Study of Language and Information, 1992.

[30] E.C.R. Hehner. do Considere od: A Contribution to the Programming Calculus. *Acta Informatica*, 11:287–304, 1979.

[31] W. H. Hesselink. *Programs, Recursion and Unbounded Choice*. Cambridge Tracts in Theoretical Computer Science 27. Cambridge University Press, 1992.

[32] C.A.R. Hoare. *Communicating Sequential Process*. Prentice-Hall International, 1985.

[33] P.-A. Masson J. Julliand and H. Mountassir. Vérification par model-checking modulaire des propriétés dynamiques introduites en B. *Technique et Science Informatique*, 20(7), 2001.

[34] D. Cansell J.-R. Abrial and D. Méry. Refinement and reachability in event_b. In Martin Henson Helen Treharne, Steve King and Steve Schneider, editors, *ZB 2005: Formal Specification and Development in Z and B, LNCS 3455*, volume 3455 of *Lecture Notes in Computer Science*, pages 222–241. Springer-Verlag, April 2005.

[35] B. Jonsson. Compositional Specification and Verification of Distributed Systems. *ACM Transactions on Programming Languages and Systems*, 16(2):259–303, March 1994.

[36] C. S. Jutla, E. Knapp, and J. R. Rao. A Predicate Transformer Approach to the Semantics of Parallel Programs. In *Proceedings of the Eight Annual ACM Symposium on the Principles of Distributed Computing*, pages 249–263, 1989.

[37] C. S. Jutla and J. R. Rao. A Methodology for Designing Proof Rules for Fair Parallel Programs. *Formal Aspects of Computing*, 9:359–378, 1997.

[38] Y. Kesten, Z. Manna, and A. Pnueli. Temporal Verification of Simulation and Refinement. In *REX scholl a decade of concurrency: Reflections and perspectives, LNCS 803*, pages 178–196. Springer-Verlag, 1994.

[39] E. Knapp. A Predicate Transformer for Progress. *Information Processing Letters*, 33:323–330, 1989.

[40] L. Lamport. The mutual exclusion problem — parts i and ii. *Journal of the ACM*, 33(2):313–348, January 1985.

[41] L. Lamport. The Temporal Logic of Actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994.

[42] Leslie Lamport and Fred B. Schneider. The "Hoare Logic" of CSP and All That. *ACM Transactions on Programming Languages and Systems*, 6(2):281–296, April 1984.

[43] Z. Manna and A. Pnueli. Completing the Temporal Picture. *Theoretical Computer Science*, 83(1):97–130, 1991.

[44] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems : Specification* . Springer, 1992.

[45] Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems: Progress.* `http://theory.stanford.edu/~zm/tvors3.html`. Draft, 1996.

[46] D. Meier and B. A. Sanders. Composing leads-to Properties. *Theoretical Computer Science*, 243:339–361, 2000.

[47] D. Méry. Machines abstraites temporelles analyse comparative de B et TLA$^+$. In *First B Conference*, pages 191–220. Nantes, november 1996.

[48] J. Misra. A logic for concurrent programming: Progress. *Journal of Computer and Software Engineering*, 3(2):273–300, 1995.

[49] J. Misra. A logic for concurrent programming: Safety. *Journal of Computer and Software Engineering*, 3(2):239–272, 1995.

[50] G. Nelson. A Generalization of Dijkstra's Calculus. *ACM Transactions on Programming Languages and Systems*, 11(4):517–561, October 1989.

[51] S. Owicki and D. Gries. An Axiomatic Proof Technique for Parallel Programs. *Acta Informatica*, 6:319–340, 1976.

[52] D. Park. A Predicate Transformer for Weak Fair Iteration. In *Proceedings of the 6th IBM Symposium on Mathematical Foundations in Computer Science, Hakone, Japan*, pages 257–275. IBM, 1981.

[53] I. S. W. B. Prasetya. Error in the UNITY Substitution Rule for Subscripted Operators. *Formal Aspects of Computing*, 6:466–470, 1994.

[54] J. R. Rao. *Extensions of the UNITY Methodology.* Number 908 in Lectures Notes in Computer Science. Springer, 1995.

[55] Glenn Ricart and Ashok K. Agrawala. An Optimal Algorithm for Mutual Exclusion in Computer Networks. *Communications of the ACM*, 24(1):9–17, 1981.

[56] H. Ruíz Barradas and D. Bert. Specification and Proof of Liveness Properties under Fairness Assumptions in B Event Systems. In *Integrated Formal Methods, Third International Conference IFM 2002, LNCS 2335*, pages 360–379. Springer-Verlag, May 2002.

[57] H. Ruíz Barradas and D. Bert. Propriétés dynamiques avec hypothèses d'équité en B événementiel. In *AFADL'2004*, pages 299–313. LIFC, Besançon, France, 2004.

[58] H. Ruíz Barradas and D. Bert. A Fixpoint Semantics of Event Systems with and without Fairness Assumptions. Technical Report 1081-I LSR 21, LSR-IMAG, Grenoble, 2005.

[59] H. Ruíz Barradas and D. Bert. Proof Obligations for Specification and Refinement of Liveness Properties under Weak Fairness. Technical Report 1071-I LSR 20, LSR-IMAG, Grenoble, 2005.

[60] H. Ruíz Barradas and D. Bert. Propriétés dynamiques avec hypothèses d'équité en B événementiel. *Technique et Science Informatique,RSTI,série TSI*, 25(1):73–102, 2006.

[61] Héctor Ruíz Barradas and Didier Bert. A Fixpoint Semantics of Event Systems with and without Fairness Assumptions. In *Fifth International Conference on Integrated Formal Methods IFM 2005, LNCS 3771*. Springer-Verlag, 2005.

[62] B. A. Sanders. Eliminating the Substitution Axiom from UNITY Logic. *Acta Informatica*, 3:189–205, 1991.

[63] B. A. Sanders. On the UNITY Design Decisions. In *Research Directions in High-Level Parallel Programming Languages , LNCS 574*, pages 50–63. Springer-Verlag, 1991.

[64] B. A. Sanders. Data Refinement of Mixed Specifications . *Acta Informatica*, 35:91–129, 1998.

[65] T. Scheurer. *Foundations of Computing System Development with Set Theory and Logic*. Addison-Wesley, 1994.

[66] E. Sedletsky, A. Pnueli, and M. Ben-Ari. Formal Verification of the Ricart-Agrawala Algorithm. In S. Kapoor and S. Prasad, editors, *Proc. of FST TCS 2000, LNCS 1974, Pise*, pages 325–335. Springer-Verlag, 2000.

[67] A. K. Sing. Program Refinement in Fair Transition Systems. *Acta Informatica*, 30:503–535, 1993.

[68] N. H. Xuong. *Mathématiques Discrètes et Informatique*. Masson, 1992.

Appendix A

Reachability Relation between Predicates or Sets

A.1 Relation *leads-to*

In order to guarantee that *leads-to* ($\leadsto$), as a relation between predicates on the system state, and the relation $\mathcal{L}_{\mathcal{E}}$ between subsets of u are equivalent, we supposed the following equivalence:

$$P(x) \leadsto Q(x) \equiv \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}} \tag{4.7}$$

In the following paragraphs we give the proof of this equivalence. It is founded on the fact that *leads-to* relation of UNITY logic, as pointed in [37], can be defined by an induction scheme, similar to definition 4.4. That is, $\leadsto$ as a relation between predicates, $\leadsto \subseteq \mathsf{Pred} \times \mathsf{Pred}$, where Pred is the set of predicates on the space of the state variable x, is the smallest relation satisfying rules BRL, TRA and DSJ given in section 3.1.2[1]:

BRL: $B \subseteq \leadsto$

TRA: $\leadsto^2 \subseteq \leadsto$

DSJ: $\forall m \cdot (m \in M \Rightarrow P(m) \leadsto Q) \Rightarrow \exists m \cdot (m \in M \wedge P(m)) \leadsto Q$

where B is the set of couples of predicates satisfying the *ensures* relation:

$$B = \{\, P \mapsto Q \mid P \gg Q \,\} \tag{A.1}$$

We recall that $\gg$ relation must be instantiated to $\gg_m$ relation under minimal progress hypothesis or $\gg_w$ relation under weak fairness assumptions, in similar way to the instantiation of the basic relation $\mathcal{E}$. In order to give the proof of equivalence (4.7), at this time we suppose that B and $\mathcal{E}$ satisfy the following properties:

$$\forall (P, Q) \cdot (P \in \mathsf{Pred} \wedge Q \in \mathsf{Pred} \Rightarrow$$
$$P(x) \gg Q(x) \equiv \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{E}) \tag{A.2}$$
$$\forall (p, q) \cdot (p \subseteq u \wedge q \subseteq u \Rightarrow p \mapsto q \in \mathcal{E} \equiv x \in p \gg x \in q) \tag{A.3}$$

[1]The infix notation $P \leadsto Q$ is used to state that $P \mapsto Q \in \leadsto$, for any predicate P and Q in Pred.

We give below the proof of these properties, instantiated to weak fairness or minimal progress assumptions.

The proof of (4.7) is given in two parts:

$$P(x) \leadsto Q(x) \Rightarrow \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}} \tag{A.4}$$

$$\{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}} \Rightarrow P(x) \leadsto Q(x) \tag{A.5}$$

Proof of (A.4)

Let $\mathcal{P}$ be the following set:

$$\mathcal{P} = \{\, P \mapsto Q \mid (P \leadsto Q) \wedge \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}} \,\}$$

From this definition follows $\mathcal{P} \subseteq \leadsto$. Inclusion $\leadsto \subseteq \mathcal{P}$ is proved below by structural induction. From these inclusions, follows the equality $\mathcal{P} = \leadsto$. Finally, from this equality follows (A.4):

$$\begin{array}{ll} & \mathcal{P} = \leadsto \\ \equiv & \{\ \mathcal{P} = \leadsto \cap \{\, P \mapsto Q \mid \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}} \,\}\ \} \\ & \leadsto \subseteq \{\, P \mapsto Q \mid \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}} \,\} \\ \equiv & \\ & \forall (P, Q) \cdot (P \leadsto Q \Rightarrow \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}}) \end{array}$$

□

In order to proof $\leadsto \subseteq \mathcal{P}$, the following proofs are required:

- $B \subseteq \mathcal{P}$.
- $\mathcal{P}^2 \subseteq \mathcal{P}$.
- $\forall m \cdot (m \in M \Rightarrow P(m) \mapsto Q \in \mathcal{P}) \Rightarrow \exists m \cdot (m \in M \wedge P(m)) \mapsto Q \in \mathcal{P}$

Proof of Base Case

1.	$P \gg Q \Rightarrow \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{E}$	; from (A.2)
2.	$P \gg Q \Rightarrow \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}}$	; 1 and def. 4.4
3.	$P \gg Q \Rightarrow P \leadsto Q$	; BRA
4.	$P \mapsto Q \in B \Rightarrow P \mapsto Q \in \mathcal{P}$	; 3, 2 and (A.1)
5.	$B \subseteq \mathcal{P}$	; 4

□

Proof of Transitivity
It follows from $P \mapsto Q \in \mathcal{P} \wedge Q \mapsto R \in \mathcal{P} \Rightarrow P \mapsto R \in \mathcal{P}$:

1.	$P \mapsto Q \in \mathcal{P} \wedge Q \mapsto r \in \mathcal{P}$	; premise
2.	$P \leadsto R$	; 1, def. $\mathcal{P}$, TRA
3.	$\{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge R(z) \,\} \in \mathcal{L}_{\mathcal{E}}$	; 1, def. $\mathcal{P}$, STR
4.	$P \mapsto R \in \mathcal{P}$	; 3, 2 and def. $\mathcal{P}$

□

Proof of Disjunction

1. $\forall m \cdot (m \in M \Rightarrow P(m) \mapsto Q \in \mathcal{P})$; premise
2. $\exists m \cdot (m \in M \wedge P(m)) \rightsquigarrow Q$; 1, def. $\mathcal{P}$, DSJ
3. $\forall m \cdot (m \in M \Rightarrow \{\, z \mid z \in u \wedge P(m)(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}})$; 1, def. $\mathcal{P}$
4. $\{\, \{\, z \mid z \in u \wedge P(m)(z) \,\} \mid m \in M \,\} \times \{\, z \mid z \in u \wedge Q(z) \,\} \subseteq \mathcal{L}_{\mathcal{E}}$; 3
5. $\bigcup(\{\, \{\, z \mid z \in u \wedge P(m)(z) \,\} \mid m \in M \,\}) \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}}$; 4, SDJ
6. $\{\, z \mid z \in u \wedge \exists m \cdot (m \in M \wedge P(m)(z)) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}}$; 5
7. $\exists m \cdot (m \in M \wedge P(m)) \mapsto Q \in \mathcal{P}$; 6, 2 and def. $\mathcal{P}$

□

Proof of (A.5)

This proof is similar to the proof of (A.4). Let $\mathcal{Q}$ be the following set:

$$\mathcal{Q} = \{\, p \mapsto q \mid p \mapsto q \in \mathcal{L}_{\mathcal{E}} \wedge x \in p \rightsquigarrow x \in q \,\}$$

From this definition follows $\mathcal{Q} \subseteq \mathcal{L}_{\mathcal{E}}$. Inclusion $\mathcal{L}_{\mathcal{E}} \subseteq \mathcal{Q}$ is proved below by structural induction. From these inclusions follows equality $\mathcal{Q} = \mathcal{L}_{\mathcal{E}}$. Now, from this equality follows inclusion $\mathcal{L}_{\mathcal{E}} \subseteq \{\, p \mapsto q \mid p \mapsto q \in \mathcal{L}_{\mathcal{E}} \wedge x \in p \rightsquigarrow x \in q \,\}$:

$$\begin{array}{ll} & \mathcal{Q} = \mathcal{L}_{\mathcal{E}} \\ \equiv & \{\ Q = \mathcal{L}_{\mathcal{E}} \cap \{\, p \mapsto q \mid p \subseteq u \wedge q \subseteq u \wedge x \in p \rightsquigarrow x \in q \,\}\ \} \\ & \mathcal{L}_{\mathcal{E}} \subseteq \{\, p \mapsto q \mid p \subseteq u \wedge q \subseteq u \wedge x \in p \rightsquigarrow x \in q \,\} \end{array}$$

Finally, from this inclusion, and taking $\{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}}$ as a premise, the conclusion $P \rightsquigarrow Q$ of (A.5) follows.

□

In order to prove $\mathcal{L}_{\mathcal{E}} \subseteq \mathcal{Q}$ the following proofs are required:

- $\mathcal{E} \subseteq \mathcal{Q}$.
- $\mathcal{Q}^2 \subseteq \mathcal{Q}$.
- $l \times \{q\} \subseteq \mathcal{Q} \Rightarrow \bigcup(l) \mapsto q \in \mathcal{Q}$

Proof of Base Case

1. $p \mapsto q \in \mathcal{E} \Rightarrow x \in p \gg x \in q$; from (A.3)
2. $p \mapsto q \in \mathcal{E} \Rightarrow x \in p \rightsquigarrow x \in q$; 1 and BRL
3. $p \mapsto q \in \mathcal{E} \Rightarrow p \mapsto q \in \mathcal{L}_{\mathcal{E}}$; SBR
4. $p \mapsto q \in \mathcal{E} \Rightarrow p \mapsto q \in \mathcal{Q}$; 3, 2, def. $\mathcal{Q}$
5. $\mathcal{E} \subseteq \mathcal{Q}$; 4

□

Proof of Transitivity

1. $p \mapsto q \in \mathcal{Q} \wedge q \mapsto r \in \mathcal{Q}$; premise
2. $p \mapsto q \in \mathcal{L}_{\mathcal{E}} \wedge q \mapsto r \in \mathcal{L}_{\mathcal{E}}$; 1, def $\mathcal{Q}$
3. $(x \in p \rightsquigarrow x \in q) \wedge (x \in q \rightsquigarrow x \in r)$; 1, def Q
4. $p \mapsto r \in \mathcal{L}_{\mathcal{E}} \wedge x \in p \rightsquigarrow x \in r$; 2, 3, TRA,STR
5. $p \mapsto r \in \mathcal{Q}$; 4, def. $\mathcal{Q}$

□

Proof of Disjunction

1.	$l \times \{q\} \subseteq \mathcal{Q}$	; premise
2.	$l \times \{q\} \subseteq \mathcal{L}_{\mathcal{E}}$	; 1, def. $\mathcal{Q}$
3.	$\bigcup(l) \mapsto q \in \mathcal{L}_{\mathcal{E}}$	; 2, SDR
4.	$l \times \{q\} \subseteq \{\, a \mapsto b \mid a \subseteq u \wedge b \subseteq u \wedge x \in a \rightsquigarrow x \in b \,\}$	; 1, def. $\mathcal{Q}$
5.	$\forall s \cdot (s \in l \Rightarrow x \in s \rightsquigarrow x \in q)$	; $l \subseteq \mathbb{P}(u)$, 4
6.	$\exists s \cdot (s \in l \wedge x \in s) \rightsquigarrow x \in q$	; 5, DSJ
7.	$x \in \bigcup(l) \rightsquigarrow x \in q$	; 6
8.	$\bigcup(l) \mapsto q \in \mathcal{Q}$	; 7, 3, def. $\mathcal{Q}$

□

A.2 Instantiation of *ensures* to Minimal Progress

In this section, properties (A.2) and (A.3) are proved when the *ensures* relation is instantiated to a minimal progress assumption.

Definition of *ensures* relation under minimal progress assumptions ($\gg_m$), without considering the strongest invariant, is given by the following definition:

$$\begin{array}{l}\forall (P,Q) \cdot (P \in \mathsf{Pred} \wedge Q \in \mathsf{Pred} \Rightarrow P(x) \gg_m Q(x) \equiv \\ \quad \forall x \cdot (I(x) \wedge P(x) \wedge \neg Q(x) \Rightarrow \mathit{grd}(\mathsf{sub}(S)) \wedge [\mathsf{sub}(S)]\, Q(x)))\end{array}$$

where $\mathsf{sub}(S)$ denotes the *generalized substitution* associated with set transformer S.

Proof of (A.2)

$$\begin{array}{ll}
& P(x) \gg_m Q(x) \\
\equiv & \qquad \{\ \text{def. } \gg_m\ \} \\
& \forall x \cdot (I(x) \wedge P(x) \wedge \neg Q(x) \Rightarrow \mathit{grd}(\mathsf{sub}(S)) \wedge [\mathsf{sub}(S)]\, Q(x)) \\
\equiv & \qquad \{\ \text{set theory}\ \} \\
& \forall x \cdot (x \in \{\, z \mid I(z) \wedge P(z) \,\} \cap \{\, z \mid I(z) \wedge \neg Q(z) \,\} \Rightarrow I(x) \wedge \mathit{grd}(\mathsf{sub}(S)) \wedge \\
& [\mathsf{sub}(S)]\, Q(x)) \\
\equiv & \qquad \{\ I(x) \Rightarrow [\mathsf{sub}(S)]\, I(x), \text{ conjunctive } S\ \} \\
& \forall x \cdot (x \in \{\, z \mid I(z) \wedge P(z) \,\} \cap \{\, z \mid I(z) \wedge \neg Q(z) \,\} \Rightarrow I(x) \wedge \mathit{grd}(\mathsf{sub}(S)) \wedge \\
& [\mathsf{sub}(S)]\, I(x) \wedge Q(x)) \\
\equiv & \qquad \{\ \text{set transformers}\ \} \\
& \forall x \cdot (x \in \{\, z \mid I(z) \wedge P(z) \,\} \cap \{\, z \mid I(z) \wedge \neg Q(z) \,\} \Rightarrow x \in \mathsf{grd}(S) \cap S(\{\, z \mid I(z) \wedge Q(z) \,\})) \\
\equiv & \qquad \{\ \text{set theory, } u = \{\, z \mid I(z) \,\}\ \} \\
& \{\, z \mid z \in u \wedge P(z) \,\} \cap \overline{\{\, z \mid z \in u \wedge Q(z) \,\}} \subseteq \mathsf{grd}(S) \cap S(\{\, z \mid z \in u \wedge Q(z) \,\}) \\
\equiv & \qquad \{\ \text{def. } \mathcal{E}_m\ \} \\
& \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{E}_m
\end{array}$$

□

Proof of (A.3)

$$
\begin{array}{ll}
& p \mapsto q \in \mathcal{E}_m \\
\equiv & \qquad \{ \text{ def. } \mathcal{E}_m \ \} \\
& p \cap \overline{q} \subseteq S(q) \cap \mathsf{grd}(S) \\
\equiv & \\
& \forall x \cdot (x \in p \cap \overline{q} \Rightarrow x \in S(q) \cap \mathsf{grd}(S)) \\
\equiv & \qquad \{ \ x \in p \Rightarrow I(x) \ \} \\
& \forall x \cdot (I(x) \wedge x \in p \wedge \neg x \in q \Rightarrow x \in S(q) \cap \mathsf{grd}(S)) \\
\equiv & \qquad \{ \text{ set transformers } \} \\
& \forall x \cdot (I(x) \wedge x \in p \wedge \neg x \in q \Rightarrow [\mathsf{sub}(S)]\, x \in q \wedge grd(S)) \\
\equiv & \qquad \{ \text{ def. } \gg_m \ \} \\
& x \in p \gg_m x \in q
\end{array}
$$

□

A.3 Instantiation of *ensures* to Weak Fairness

In this section, properties (A.2) and (A.3) are proved when the *ensures* relation ($\gg$) is instantiated to a weak fairness assumption relation ($\gg_w$). The instantiation under this condition is given by the following equivalence:

$$P(x) \gg Q(x) \equiv \exists G \cdot (G \in E \wedge G \cdot P(x) \gg_w Q(x))$$

Definition of *ensures* relation under weak fairness assumptions ($\gg_w$), without considering the strongest invariant, is given by the following definition:

$$
\begin{array}{l}
\forall (P, Q) \cdot (P \in \mathsf{Pred} \wedge Q \in \mathsf{Pred} \Rightarrow G \cdot P(x) \gg_w Q(x) \equiv \\
\quad \forall x \cdot (I(x) \wedge P(x) \wedge \neg Q(x) \Rightarrow ([\mathsf{sub}(S)]\, (P(x) \vee Q(x))) \wedge grd(\mathsf{sub}(G)) \wedge [\mathsf{sub}(G)]\, Q(x)))
\end{array}
$$

where $\mathsf{sub}(S)$ and $\mathsf{sub}(G)$ denote the *generalized substitutions* associated with set transformers S and G.

The proof follows from the following equivalences which are proved below:

$$G \cdot P(x) \gg_w Q(x) \equiv \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{E}(G) \tag{A.6}$$

$$p \mapsto q \in \mathcal{E}(G) \equiv G \cdot x \in p \gg_w x \in q \tag{A.7}$$

Proof of (A.2)
Implication from left to right follows from (A.6)

$$
\begin{array}{ll}
& G \cdot P(x) \gg_w Q(x) \equiv \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{E}(G) \\
\Rightarrow & \qquad \{ \ \mathcal{E}_w = \bigcup G \cdot (G \in E \mid \mathcal{E}(G)) \ \} \\
& G \cdot P(x) \gg_w Q(x) \Rightarrow \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{E}_w \\
\Rightarrow & \\
& \exists G \cdot (G \in E \wedge G \cdot P(x) \gg_w Q(x)) \Rightarrow \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{E}_w
\end{array}
$$

Implication from right to left follows from (A.6)

$$
\begin{array}{ll}
& G \cdot P(x) \gg_w Q(x) \equiv \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{E}(G) \\
\Rightarrow & \\
& \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{E}(G) \Rightarrow \exists G \cdot (G \in E \wedge G \cdot P(x) \gg_w Q(x)) \\
\Rightarrow & \\
& \exists G \cdot (G \in E \wedge \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{E}(G)) \Rightarrow \exists G \cdot (G \in E \wedge G \cdot P(x) \gg_w Q(x)) \\
\Rightarrow & \\
& \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \bigcup G \cdot (G \in E \mid \mathcal{E}(G)) \Rightarrow \exists G \cdot (G \in E \wedge G \cdot P(x) \gg_w Q(x)) \\
\equiv & \qquad \{\ \text{def. } \mathcal{E}\ \} \\
& \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(z) \,\} \in \mathcal{E}_w \Rightarrow \exists G \cdot (G \in E \wedge G \cdot P(x) \gg_w Q(x))
\end{array}
$$

□

Proof of (A.3)
Implication from left to right follows from (A.7)

$$
\begin{array}{ll}
& p \mapsto q \in \mathcal{E}(G) \equiv G \cdot x \in p \gg_w x \in q \\
\Rightarrow & \\
& p \mapsto q \in \mathcal{E}(G) \Rightarrow \exists G \cdot (G \in E \wedge G \cdot x \in p \gg_w x \in q) \\
\Rightarrow & \\
& \exists G \cdot (G \in E \wedge p \mapsto q \in \mathcal{E}(G)) \Rightarrow \exists G \cdot (G \in E \wedge G \cdot x \in p \gg_w x \in q) \\
\Rightarrow & \qquad \{\ \mathcal{E}_w = \bigcup G \cdot (G \in E \mid \mathcal{E}(G))\ \} \\
& p \mapsto q \in \mathcal{E}_w \Rightarrow \exists G \cdot (G \in E \wedge G \cdot x \in p \gg_w x \in q)
\end{array}
$$

Implication from right to left follows from (A.7)

$$
\begin{array}{ll}
& p \mapsto q \in \mathcal{E}(G) \equiv G \cdot x \in p \gg_w x \in q \\
\Rightarrow & \\
& G \cdot x \in p \gg_w x \in q \Rightarrow p \mapsto q \in \mathcal{E}(G) \\
\Rightarrow & \qquad \{\ \mathcal{E}_w = \bigcup G \cdot (G \in E \mid \mathcal{E}(G))\ \} \\
& G \cdot x \in p \gg_w x \in q \Rightarrow p \mapsto q \in \mathcal{E}_w \\
\equiv & \\
& \exists G \cdot (G \in E \wedge G \cdot x \in p \gg_w x \in q) \Rightarrow p \mapsto q \in \mathcal{E}_w
\end{array}
$$

□

Proof of (A.6)

$$
\begin{array}{ll}
& G \cdot P(x) \gg_w Q(x) \\
\equiv & \{\ \text{def. } \gg_w\ \} \\
& \forall x \cdot (I(x) \ \wedge\ P(x) \ \wedge\ \neg Q(x) \ \Rightarrow\ ([\mathsf{sub}(S)]\,(P(x) \ \vee\ Q(x))) \ \wedge\ \mathit{grd}(\mathsf{sub}(G)) \ \wedge \\
& [\mathsf{sub}(G)]\,Q(x)) \\
\equiv & \{\ \text{set theory}\ \} \\
& \forall x \cdot (x \in \{\, z \mid I(z) \ \wedge\ P(z) \,\} \cap \{\, z \mid I(z) \ \wedge\ \neg Q(x) \,\} \ \Rightarrow\ ([\mathsf{sub}(S)]\,(P(x) \ \vee\ Q(x))) \ \wedge \\
& \mathit{grd}(\mathsf{sub}(G)) \wedge [\mathsf{sub}(G)]\,Q(x)) \\
\equiv & \{\ I(x) \Rightarrow [\mathsf{sub}(S)]\,I(x), \text{ conjunctive } S \text{ and } G\ \} \\
& \forall x \cdot (x \in \{\, z \mid I(z) \wedge P(z) \,\} \cap \{\, z \mid I(z) \wedge \neg Q(x) \,\} \Rightarrow ([\mathsf{sub}(S)]\,(I(x) \wedge P(x) \vee I(x) \wedge \\
& Q(x))) \wedge \mathit{grd}(\mathsf{sub}(G)) \wedge [\mathsf{sub}(G)]\,I(x) \wedge Q(x)) \\
\equiv & \{\ \text{set transformers}\ \} \\
& \forall x \cdot (x \in \{\, z \mid I(z) \ \wedge\ P(z) \,\} \cap \{\, z \mid I(z) \ \wedge\ \neg Q(x) \,\} \ \Rightarrow\ x \in S(\{\, z \mid z \in u \ \wedge\ P(z) \,\} \cup \\
& \{\, z \mid z \in u \wedge Q(x) \,\}) \wedge x \in \mathsf{grd}(G) \wedge x \in G(\{\, z \mid z \in u \wedge Q(x) \,\})) \\
\equiv & \{\ \text{set theory, } u = \{\, z \mid I(z) \,\}\ \} \\
& \{\, z \mid z \ \in\ u \ \wedge\ P(z) \,\} \cap \overline{\{\, z \mid z \in u \wedge Q(x) \,\}} \ \subseteq\ S(\{\, z \mid z \ \in\ u \ \wedge\ P(z) \,\} \cup \{\, z \mid z \ \in\ u \ \wedge \\
& Q(x) \,\}) \cap \mathsf{grd}(G) \cap G(\{\, z \mid z \in u \wedge Q(x) \,\}) \\
\equiv & \{\ \text{def. } \mathcal{E}(G)\ \} \\
& \{\, z \mid z \in u \wedge P(z) \,\} \mapsto \{\, z \mid z \in u \wedge Q(x) \,\} \in \mathcal{E}(G)
\end{array}
$$

□

Proof of (A.7)

$$
\begin{array}{ll}
& p \mapsto q \in \mathcal{E}(G) \\
\equiv & \{\ \text{def. } \mathcal{E}_w\ \} \\
& p \cap \overline{q} \subseteq S(p \cup q) \cap \mathsf{grd}(G) \cap G(q) \\
\equiv & \\
& \forall x \cdot (x \in p \cap \overline{q} \Rightarrow x \in S(p \cup q) \cap \mathsf{grd}(G) \cap G(q)) \\
\equiv & \{\ x \in p \Rightarrow I(x)\ \} \\
& \forall x \cdot (I(x) \wedge x \in p \wedge \neg x \in q \Rightarrow x \in S(p \cup q) \cap \mathsf{grd}(G) \cap G(q)) \\
\equiv & \{\ \text{set transformers}\ \} \\
& \forall x \cdot (I(x) \wedge x \in p \wedge \neg x \in q \Rightarrow ([\mathsf{sub}(S)]\,(x \in p \vee x \in q)) \wedge \mathit{grd}(G) \wedge [\mathsf{sub}(G)]\,x \in q) \\
\equiv & \{\ \text{def. } \gg_w\ \} \\
& G \cdot x \in p \gg_w x \in q
\end{array}
$$

□

Appendix B

Extending the Semantics to Consider the Strongest Invariant

In this appendix, the strongest invariant is considered in definitions of *termination* and *reachability* relations. It allows us to preserve soundness of UNITY logic. Certain definitions and proofs are independent of the strongest invariant and they remain unchanged. New definitions and proofs are only considered in this appendix. It is structured in four parts. In section B.1, the strongest invariant is presented. In section B.2, the general theorem about soundness and completeness is restated and proved. In section B.4, *termination* and basic relation for *leads-to* are redefined to consider a minimal progress assumption, and the hypothesis of the theorem of soundness and completeness are proved. In section B.5 an similar treatment is considered to the case of a weak fairness assumption.

B.1 Strongest Invariant

Original definitions of the fundamental relations *unless* and *ensures* in UNITY logic [21], do not consider initial conditions. On another hand, in order to give the possibility to prove valid properties (completeness), in [21] the *Substitution Axiom* is proposed. Basically, this axiom states that any invariant predicate can be replaced by *true* and vice versa.

Original definitions of the fundamental relations in [21], along with the substitution axiom, give an unsound proof system as it is reported in [62]. To fix this problem, the relation *unless* and *ensures* are redefined to consider initial conditions. The new definitions in [62] consider the *strongest invariant*, which holds in the reachable states. Moreover, the substitution axiom is replaced by a substitution rule, which becomes a theorem in the new logic. However, in [53], a problem with the new rule is reported and another substitution rule is proposed. In this report we do not consider this last issue.

Following the proposal in [62], and adapting the definition to our framework, the strongest invariant is defined as follows:

Definition B.1 *Strongest Invariant*

Let SS be a B *event system with state variable x, initialization U and choice of events S. The strongest invariant SI of SS is the strongest predicate X satisfying:*

$$\forall x \cdot ((X \Rightarrow [S]\, X) \wedge (([x' := x]\, \mathit{prd}(U)) \Rightarrow X))$$

where $\mathsf{prd}(U)$ denotes the before-after relation associated with the initilization U [2]. Using the strongest invariant, the definition of the *ensures* relation, to specify basic liveness properties, under our two fairness assumptions is as follows:

Definition B.2 *Ensures under Minimal Progress*

Let SS be a B *event system with state variable x, initialization U and choice of events S. For any predicate P and Q over x, the basic liveness relation $\gg_m$ is defined as follows:*

$$P \gg_m Q \equiv SI \wedge P \wedge \neg Q \Rightarrow (([S]\ Q) \wedge grd(S))$$

Definition B.3 *Ensures under Weak Fairness*

Let SS be a B *event system with state variable x, initialization U, choice of events S and G an event of SS. For any predicate P and Q over x, the basic liveness relation $\gg_w$ is defined as follows:*

$$G \cdot P \gg_w Q \equiv SI \wedge P \wedge \neg Q \Rightarrow (([S]\ P \vee Q) \wedge ([G]\ Q) \wedge grd(S))$$

Finally si, the set counterpart of the strongest invariant, is given by the following definition

$$si = \{z \mid SI(z)\}$$

B.2 General Framework

The body of iteration in the general framework does not change:

$$\mathcal{F}(r) = (\overline{r} \Longrightarrow W) \tag{4.4}$$

The *termination* relation is redefined to consider the strongest invariant as follows:

Definition B.4 (Termination Relation)

$$\mathcal{T} = \{\, a \mapsto b \mid a \subseteq u \wedge b \subseteq u \wedge si \cap a \subseteq \mathsf{fix}(\mathcal{F}(si \cap b)) \,\} \tag{B.1}$$

The *reachability* relation is not modified directly:

Definition B.5 (Reachability Relation)
The reachability *relation $\mathcal{L}_\mathcal{E}$, $\mathcal{L}_\mathcal{E} \in \mathbb{P}(u) \leftrightarrow \mathbb{P}(u)$, is defined by the following induction scheme:*
(**SBR**)*: $\mathcal{E} \subseteq \mathcal{L}_\mathcal{E}$*
(**STR**)*: $\mathcal{L}_\mathcal{E}$; $\mathcal{L}_\mathcal{E} \subseteq \mathcal{L}_\mathcal{E}$*
(**SDR**)*: $\forall (q, l) \cdot (q \in \mathbb{P}(u) \wedge l \subseteq \mathbb{P}(u) \Rightarrow (l \times \{q\} \subseteq \mathcal{L}_\mathcal{E} \Rightarrow \bigcup(l) \mapsto q \in \mathcal{L}_\mathcal{E}))$*
Closure*: $\forall l' \cdot (l' \in u \leftrightarrow u \wedge \mathcal{E} \subseteq l' \wedge l' ; l' \subseteq l' \wedge$*
$\forall (q, l) \cdot (q \in \mathbb{P}(u) \wedge l \subseteq \mathbb{P}(u) \wedge l \times \{q\} \subseteq l' \Rightarrow \bigcup(l) \mapsto q \in l') \Rightarrow \mathcal{L}_\mathcal{E} \subseteq l')$

However, when $\mathcal{L}_\mathcal{E}$ is instantiated to minimal or weak fairness assumptions, the strongest invariant is considered in definition of the base relation.

The theorem of soundness and completeness, taking into account new definitions of $\mathcal{T}$ and $\mathcal{L}_\mathcal{E}$ remains basically unchanged. Only hypothesis concerning the strongest invariant and the basic relation, are modified with respect to the precedent version.

Theorem 6 (Soundness and Completeness)
Let W be a monotonic and strict set transformer and $\mathcal{F}(r) = (\overline{r} \Longrightarrow W)$ for any r in $\mathbb{P}(u)$. Let relations $\mathcal{T}$ and $\mathcal{L}_{\mathcal{E}}$ be defined as definitions B.4 and B.5 respectively. Considering (a) $a \mapsto b \in \mathcal{E} \Rightarrow si \cap a \cap \overline{b} \subseteq W(si \cap b)$, (b) $a \subseteq b \Rightarrow a \mapsto b \in \mathcal{E}$, (c) $si \cap a \subseteq b \Rightarrow a \mapsto b \in \mathcal{E}$ and (d) $W(r) \mapsto r \in \mathcal{L}_{\mathcal{E}}$, for any a, b and r in $\mathbb{P}(u)$, the following equality holds:

$$\mathcal{L}_{\mathcal{E}} = \mathcal{T}$$

The prove is given in the following section.

In appendix A, the equivalence between the reachability relation $\mathcal{L}_{\mathcal{E}}$ and the definition of the *leads-to* relation $\rightsquigarrow$ is proved. That proof does not consider the strongest invariant in the given definitions. In order to connect the definition of the reachability relation, considering the strongest invariant, a demonstration similar to the proof of (4.7), can be given to prove the following equivalence:

$$P(x) \rightsquigarrow Q(x) \equiv \{\, z \,|\, z \in si \wedge P(z) \,\} \mapsto \{\, z \,|\, z \in si \wedge Q(z) \,\} \in \mathcal{L}_{\mathcal{E}}$$

which relates the definition of the *leads-to* relation with the reachability relation considering the strongest invariant.

B.3 Proof of Soundness and Completeness

As before, the proof is divided in $\mathcal{L}_{\mathcal{E}} \subseteq \mathcal{T}$ and $\mathcal{T} \subseteq \mathcal{L}_{\mathcal{E}}$.

B.3.1 Proof of $\mathcal{L}_{\mathcal{E}} \subseteq \mathcal{T}$

The proof of $\mathcal{L}_{\mathcal{E}} \subseteq \mathcal{T}$ follows from the closure clause in definition of $\mathcal{L}_{\mathcal{E}}$. According to this clause, $\mathcal{T}$ must contain the base relation $\mathcal{E}$, and it must be transitive and disjunctive. The following paragraphs present the proof of these cases.

Base Case

$$a \mapsto b \in \mathcal{E} \Rightarrow a \mapsto b \in \mathcal{T} \tag{B.2}$$

1.	$a \mapsto b \in \mathcal{E}$	; premise
2.	$si \cap a \cap \overline{b} \subseteq W(b \cap si)$	; 1 and hyp. (a)
3.	$si \cap a \subseteq b \cup W(b \cap si)$	; 2
4.	$si \cap a \subseteq si \cap b \cup W(b \cap si)$	; 3
5.	$si \cap a \subseteq \mathcal{F}(si \cap b)(b \cap si)$	; 4 and def. (4.4)
6.	$si \cap a \subseteq \mathcal{F}(si \cap b)^2$	; 5 and iterate (4.5)
7.	$si \cap a \subseteq \mathsf{fix}(\mathcal{F}(si \cap b))$	; 6 and (4.8)
8.	$a \mapsto b \in \mathcal{T}$	; 7 and def. (B.1)

□

Transitivity

$$\forall(a,b)\cdot(a\mapsto b\in(\mathcal{T}\ ;\ \mathcal{T})\Rightarrow a\mapsto b\in\mathcal{T}) \tag{B.3}$$

The proof of (B.3) requires the following property:

$$\forall(a,b)\cdot(a\mapsto b\in\mathcal{T}\Rightarrow \mathsf{fix}(\mathcal{F}(si\cap a))\subseteq \mathsf{fix}(\mathcal{F}(si\cap b))) \tag{B.4}$$

Taking $a\mapsto b\in\mathcal{T}$ as a premise, and considering $\mathsf{fix}(\mathcal{F}(si\cap a))$ as the least fixpoint of $\mathcal{F}(si\cap a)$, (B.4) follows from $\mathcal{F}(si\cap a)(\mathsf{fix}(\mathcal{F}(si\cap b)))\subseteq\mathsf{fix}(\mathcal{F}(si\cap b))$ as follows:

1. $si\cap a\subseteq\mathsf{fix}(\mathcal{F}(si\cap b))$; from $a\mapsto b\in\mathcal{T}$
2. $\mathcal{F}(si\cap b)(\mathsf{fix}(\mathcal{F}(si\cap b)))=\mathsf{fix}(\mathcal{F}(si\cap b))$; fixpoint definition
3. $W(\mathsf{fix}(\mathcal{F}(si\cap b)))\subseteq\mathsf{fix}(\mathcal{F}(si\cap b))$; 2 and (4.4)
4. $\mathcal{F}(si\cap a)(\mathsf{fix}(\mathcal{F}(si\cap b)))\subseteq\mathsf{fix}(\mathcal{F}(si\cap b))$; 3, 1 and (4.4)

□

Proof of (B.3)

1. $\exists c\cdot(a\mapsto c\in\mathcal{T}\wedge c\mapsto b\in\mathcal{T})$; from $a\mapsto b\in\mathcal{T};\mathcal{T}$
2. $\exists c\cdot(si\cap a\subseteq\mathsf{fix}(\mathcal{F}(si\cap c))\wedge c\mapsto b\in\mathcal{T})$; 3 and def. $\mathcal{T}$
3. $\exists c\cdot(si\cap a\subseteq\mathsf{fix}(\mathcal{F}(si\cap c))\wedge\mathsf{fix}(\mathcal{F}(si\cap c))\subseteq\mathsf{fix}(\mathcal{F}(si\cap b)))$; 2 and (B.4)
4. $si\cap a\subseteq\mathsf{fix}(\mathcal{F}(si\cap b))$; 3
5. $a\mapsto b\in\mathcal{T}$; 6 and def. $\mathcal{T}$

□

Disjunction

$$\forall(l,q)\cdot(l\times\{q\}\subseteq\mathcal{T}\Rightarrow\bigcup(l)\mapsto q\in\mathcal{T}) \tag{B.5}$$

1. $l\times\{q\}\subseteq\mathcal{T}$; premise
2. $\forall p\cdot(p\in l\Rightarrow p\mapsto q\in\mathcal{T})$; 1
3. $\forall p\cdot(p\in l\Rightarrow si\cap p\subseteq\mathsf{fix}(\mathcal{F}(si\cap q)))$; 2 and def. $\mathcal{T}_m$
4. $\bigcup p\cdot(p\in l\mid si\cap p)\subseteq\mathsf{fix}(\mathcal{F}(si\cap q))$; 3
5. $si\cap\bigcup p\cdot(p\in l\mid p)\subseteq\mathsf{fix}(\mathcal{F}(si\cap q))$; 4
6. $si\cap\bigcup(l)\subseteq\mathsf{fix}(\mathcal{F}(si\cap q))$; 5
7. $\bigcup(l)\mapsto q\in\mathcal{T}$; 4 and def. $\mathcal{T}_m$

□

B.3.2 Proof of $\mathcal{T}\subseteq\mathcal{L}_{\mathcal{E}}$

The proof of this inclusion requires the following property which is already proved:

$$\forall r\cdot(r\in\mathbb{P}(u)\Rightarrow\mathcal{F}(r)^{\alpha}\mapsto r\in\mathcal{L}_{\mathcal{E}}) \tag{4.10}$$

Proof of $\mathcal{T} \subseteq \mathcal{L}_{\mathcal{E}}$

1. $si \cap a \subseteq \mathsf{fix}(\mathcal{F}(si \cap b))$; from $a \mapsto b \in \mathcal{T}$
2. $\exists \alpha \cdot (si \cap a \subseteq \mathcal{F}(si \cap b)^{\alpha})$; 1 and theorem 1
3. $\exists \alpha \cdot (a \mapsto \mathcal{F}(si \cap b)^{\alpha} \in \mathcal{E})$; 2 and hyp. (c)
4. $\exists \alpha \cdot (a \mapsto \mathcal{F}(si \cap b)^{\alpha} \in \mathcal{L}_{\mathcal{E}})$; 3 and STR
5. $a \mapsto si \cap b \in \mathcal{L}_{\mathcal{E}}$; 4 and (4.10)
6. $a \mapsto b \in \mathcal{L}_{\mathcal{E}}$; 5, $si \cap b \mapsto b \in \mathcal{L}_{\mathcal{E}}$, STR

□

B.4 Minimal Progress

The body of iteration under a minimal progress assumption does not change:

$$\mathcal{F}_m(r) = (\overline{r} \Longrightarrow W_m)$$

where W_m remains defined as before:

$$W_m = \mathsf{grd}(S) \mid S \tag{4.11}$$

termination relation under weak fairness assumption is defined as follows:

$$\mathcal{T}_m = \{\, a \mapsto b \,|\, a \subseteq u \wedge b \subseteq u \wedge si \cap a \subseteq \mathsf{fix}(\mathcal{F}_m(si \cap b)) \,\} \tag{B.6}$$

Basic relation $\mathcal{E}_m$ for *reachability* relation $\mathcal{L}_{\mathcal{E}_m}$ under minimal progress assumptions is defined as follows:

$$\mathcal{E}_m = \{\, a \mapsto b \,|\, a \subseteq u \wedge b \subseteq u \wedge si \cap a \cap \overline{b} \subseteq S(b) \cap \mathsf{grd}(S) \,\} \tag{B.7}$$

Relation $\mathcal{L}_{\mathcal{E}_m}$ is defined by an induction scheme, according to definition 4.4.

The following proofs of hypothesis in theorem 6 allow us to conclude the equality between *reachability* and *termination* relations under a minimal progress assumption:

$$\mathcal{T}_m = \mathcal{L}_{\mathcal{E}_m}$$

B.4.1 Proof of Premise (a)

$$a \mapsto b \in \mathcal{E}_m \Rightarrow si \cap a \cap \overline{b} \subseteq W_m(si \cap b) \tag{B.8}$$

$$\begin{array}{ll}
& a \mapsto b \in \mathcal{E}_m \\
\equiv & \qquad \{\ \text{(B.7)}\ \} \\
& si \cap a \cap \overline{b} \subseteq S(b) \cap \mathsf{grd}(S) \\
\Rightarrow & \qquad \{\ si \subseteq S(si), \text{ conjunctive } S\ \} \\
& si \cap a \cap \overline{b} \subseteq S(si \cap b) \cap \mathsf{grd}(S) \\
\equiv & \qquad \{\ \text{(4.11)}\ \} \\
& si \cap a \cap \overline{b} \subseteq W_m(si \cap b)
\end{array}$$

□

B.4.2 Proof of Premise (b)

$$a \subseteq b \Rightarrow a \mapsto b \in \mathcal{E}_m \tag{B.9}$$

1. $a \subseteq b$; premise
2. $si \cap a \cap \overline{b} = \varnothing$; 1
3. $si \cap a \cap \overline{b} \subseteq \mathsf{grd}(S) \cap S(b)$; 2
4. $a \mapsto b \in \mathcal{E}_m$; 3 and (B.7)

□

B.4.3 Proof of Premise (c)

$$si \cap a \subseteq b \Rightarrow a \mapsto b \in \mathcal{E}_m \tag{B.10}$$

1. $si \cap a \subseteq b$; premise
2. $si \cap b \cap \overline{b} = \varnothing$; 1
3. $si \cap a \cap \overline{b} \subseteq \mathsf{grd}(S) \cap S(b)$; 2
4. $a \mapsto b \in \mathcal{E}_m$; 3 and (B.7)

□

B.4.4 Proof of Premise (d)

$$W_m(r) \mapsto r \in \mathcal{L}_{\mathcal{E}_m} \tag{B.11}$$

1. $si \cap \mathsf{grd}(S) \cap S(r) \cap \overline{r} \subseteq \mathsf{grd}(S) \cap S(r)$; trivial
2. $\mathsf{grd}(S) \cap S(r) \mapsto r \in \mathcal{E}_m$; 1 and def. $\mathcal{E}_m$
3. $W_m(r) \mapsto r \in \mathcal{E}_m$; 2 and (4.11)
4. $W_m(r) \mapsto r \in \mathcal{L}_{\mathcal{E}_m}$; 3 and def. $\mathcal{L}_{\mathcal{E}_m}$

□

B.5 Weak Fairness

The body of iteration under weak fairness assumption does not change:

$$\mathcal{F}_w(r) = \overline{r} \Longrightarrow W_w \tag{4.22}$$

where W_w is defined as follows:

$$W_w = \lambda r \cdot (r \subseteq u \mid \bigcup G \cdot (G \in E \mid Y(r)(G)(r))) \tag{4.21}$$

and $Y(r)(G)(r)$ is defined by:

$$Y(q)(G)(r) = \mathsf{FIX}(\overline{q} \Longrightarrow (\mathsf{grd}(G) \cap G(r) \mid S)) \tag{4.20}$$

termination relation under weak fairness assumption is defined as follows:

$$\mathcal{T}_w = \{\, a \mapsto b \,|\, a \subseteq u \wedge b \subseteq u \wedge si \cap a \subseteq \mathsf{fix}(\mathcal{F}_w(si \cap b)) \,\} \tag{B.12}$$

Basic relation $\mathcal{E}_w$ for *reachability* relation $\mathcal{L}_{\mathcal{E}_w}$ under weak fairness assumption is not changed directly:

$$\mathcal{E}_w = \bigcup G \cdot (G \in E \mid \mathcal{E}(G)) \tag{4.25}$$

However, $\mathcal{E}(G)$ is redefined to consider the strongest invariant:

$$\mathcal{E}(G) = \{\, a \mapsto b \mid a \subseteq u \wedge b \subseteq u \wedge si \cap a \cap \overline{b} \subseteq S(a \cup b) \cap \overline{G(\varnothing)} \cap G(b) \,\} \tag{B.13}$$

Relation $\mathcal{L}_{\mathcal{E}_w}$ is defined by an induction scheme, according to definition 4.4.

The following proofs of hypothesis in theorem 6 allow us to conclude the equality between *reachability* and *termination* relations under a weak fairness assumption:

$$\mathcal{T}_w = \mathcal{L}_{\mathcal{E}_w}$$

B.5.1 Proof of Premise (a)

$$a \mapsto b \in \mathcal{E}_w \Rightarrow si \cap a \cap \overline{b} \subseteq W_w(si \cap b) \tag{B.14}$$

The proof requires the following property:

$$\forall G \cdot (G \in E \wedge a \mapsto b \in \mathcal{E}(G) \Rightarrow si \cap a \subseteq Y(si \cap b)(G)(si \cap b)) \tag{B.15}$$

$$\begin{array}{ll}
& a \mapsto b \in \mathcal{E}(G) \\
\equiv & \qquad \{\ \text{(B.13)}\ \} \\
& si \cap a \cap \overline{b} \subseteq S(a \cup b) \cap \overline{G(\varnothing)} \cap G(b) \\
\Rightarrow & \qquad \{\ si \subseteq S(si), \text{ conjunctive } S \text{ and } G\ \} \\
& si \cap a \cap \overline{b} \subseteq S(si \cap a \cup si \cap b) \cap \overline{G(\varnothing)} \cap G(si \cap b) \\
\Rightarrow & \\
& si \cap a \subseteq si \cap b \cup S(si \cap a \cup si \cap b) \cap \overline{G(\varnothing)} \cap G(si \cap b) \\
\Rightarrow & \\
& si \cap a \cup si \cap b \subseteq si \cap b \cup S(si \cap a \cup si \cap b) \cap \overline{G(\varnothing)} \cap G(si \cap b) \\
\equiv & \qquad \{\ \text{set transformers}\ \} \\
& si \cap a \cup si \cap b \subseteq (\overline{si \cap b} \Longrightarrow \mathsf{grd}(G) \cap G(si \cap b) \mid S)(si \cap a \cup si \cap b) \\
\Rightarrow & \qquad \{\ \text{greatest fixpoint property}\ \} \\
& si \cap a \subseteq \mathsf{FIX}(\overline{si \cap b} \Longrightarrow \mathsf{grd}(G) \cap G(si \cap b) \mid S) \\
\equiv & \qquad \{\ \text{(4.20)}\ \} \\
& si \cap a \subseteq Y(si \cap b)(G)(si \cap b)
\end{array}$$

□

Proof of (B.14)

1. $\forall G \cdot (G \in E \Rightarrow (a \mapsto b \in \mathcal{E}(G) \Rightarrow si \cap a \subseteq Y(si \cap b)(G)(si \cap b)))$; (B.15)
2. $\exists G \cdot (G \in E \wedge a \mapsto b \in \mathcal{E}(G)) \Rightarrow si \cap a \subseteq W_w(si \cap b)$; 1 and (4.21).
3. $a \mapsto b \in \bigcup G \cdot (G \in E \mid \mathcal{E}(G)) \Rightarrow si \cap a \subseteq W_w(si \cap b)$; 2
4. $a \mapsto b \in \mathcal{E}_w \Rightarrow si \cap a \subseteq W_w(si \cap b)$; 3 and (4.25)
5. $a \mapsto b \in \mathcal{E}_w \Rightarrow si \cap a \subseteq si \cap b \cup W_w(si \cap b)$; 4 and $si \cap b \subseteq W_w(si \cap b)$
6. $a \mapsto b \in \mathcal{E}_w \Rightarrow si \cap a \cap \overline{b} \subseteq W_w(si \cap b)$; 5

□

B.5.2 Proof of Premise (b)

$$a \subseteq b \Rightarrow a \mapsto b \in \mathcal{E}_w \tag{B.16}$$

1. $a \subseteq b$; premise
2. $si \cap a \cap \overline{b} = \varnothing$; 1
3. $\forall G \cdot (G \in E \Rightarrow si \cap a \cap \overline{b} \subseteq S(a \cup b) \cap \overline{G(\varnothing)} \cap G(b))$; 2
4. $\forall G \cdot (G \in E \Rightarrow a \mapsto b \in \mathcal{E}(G))$; 3 and (B.13)
5. $a \mapsto b \in \bigcup G \cdot (G \in E \mid \mathcal{E}(G))$; 4
6. $a \mapsto b \in \mathcal{E}_w$; 5 (4.25)

□

B.5.3 Proof of Premise (c)

$$si \cap a \subseteq b \Rightarrow a \mapsto b \in \mathcal{E}_w \tag{B.17}$$

1. $si \cap a \subseteq b$; premise
2. $si \cap a \cap \overline{b} = \varnothing$; 1
3. $\forall G \cdot (G \in E \Rightarrow si \cap a \cap \overline{b} \subseteq S(a \cup b) \cap \overline{G(\varnothing)} \cap G(b))$; 2
4. $\forall G \cdot (G \in E \Rightarrow a \mapsto b \in \mathcal{E}(G))$; 3 and (B.13)
5. $a \mapsto b \in \bigcup G \cdot (G \in E \mid \mathcal{E}(G))$; 4
6. $a \mapsto b \in \mathcal{E}_w$; 5 (4.25)

□

B.5.4 Proof of Premise (d)

$$\forall r \cdot (r \subseteq u \Rightarrow W_w(r) \mapsto r \in \mathcal{L}_{\mathcal{E}_w}) \tag{B.18}$$

The proof requires the following property:

$$\forall (G, r) \cdot (G \in E \wedge r \subseteq u \Rightarrow Y(r)(G)(r) \mapsto r \in \mathcal{E}(G)) \tag{B.19}$$

1. $S(Y(r)(G)(r)) \subseteq S(Y(r)(G)(r) \cup r)$; monotony of S
2. $Y(r)(G)(r) = r \cup \mathsf{grd}(G) \cap G(r) \cap S(Y(r)(G)(r))$; (4.20)
3. $Y(r)(G)(r) \cap \overline{r} \subseteq \mathsf{grd}(G) \cap G(r) \cap S(Y(r)(G)(r) \cup r)$; 2, 1
4. $si \cap Y(r)(G)(r) \cap \overline{r} \subseteq Y(r)(G)(r) \cap \overline{r}$; trivial
5. $Y(r)(G)(r) \mapsto r \in \mathcal{E}(G)$; 4, 3 and (4.25)

□

Proof of (B.18)

1. $\forall G \cdot (G \in E \Rightarrow Y(r)(G)(r) \mapsto r \in \mathcal{E}(G))$; from (B.19)
2. $\forall G \cdot (G \in E \Rightarrow \mathcal{E}(G) \subseteq \mathcal{E}_w)$; def. $\mathcal{E}_w$
3. $\forall G \cdot (G \in E \Rightarrow Y(r)(G)(r) \mapsto r \in \mathcal{E}_w)$; 2 and 1
4. $\forall G \cdot (G \in E \Rightarrow Y(r)(G)(r) \mapsto r \in \mathcal{L}_{\mathcal{E}_w})$; def. $\mathcal{L}_{\mathcal{E}_w}$
5. $\{ Y(r)(G)(r) \mid G \in E \} \times \{r\} \subseteq \mathcal{L}_{\mathcal{E}_w}$; 4
6. $\bigcup(\{ Y(r)(G)(r) \mid G \in E \}) \mapsto r \in \mathcal{L}_{\mathcal{E}_w}$; 5 and SDR
7. $\bigcup G \cdot (G \in E \mid Y(r)(G)(r)) \mapsto r \in \mathcal{L}_{\mathcal{E}_w}$; 6
8. $W_w(r) \mapsto r \in \mathcal{L}_{\mathcal{E}_w}$; 7 and def. F

□

Appendix C

Proofs of Chapter 4

C.1 Proofs of the General Framework

C.1.1 Proof of (4.8): $\forall\alpha \cdot (f^\alpha \subseteq \mathbf{fix}(f))$

successor ordinal:

$$
\begin{array}{ll}
& f^\alpha \subseteq \mathsf{fix}(f) \\
\Rightarrow & \\
& f(f^\alpha) \subseteq f(\mathsf{fix}(f)) \\
\Rightarrow & \\
& f^{\alpha+1} \subseteq \mathsf{fix}(f)
\end{array}
$$

limit ordinal:

$$
\begin{array}{lll}
& \forall\beta \cdot (\beta < \alpha \Rightarrow f^\beta \subseteq \mathsf{fix}(f)) & \\
\Rightarrow & & \{\ \text{monotony}\ \} \\
& \forall\beta \cdot (\beta < \alpha \Rightarrow f(f^\beta) \subseteq f(\mathsf{fix}(f))) & \\
\Rightarrow & & \{\ \text{fixpoint def.}\ \} \\
& \forall\beta \cdot (\beta < \alpha \Rightarrow f(f^\beta) \subseteq \mathsf{fix}(f)) & \\
\Rightarrow & & \\
& \bigcup\beta \cdot (\beta < \alpha \mid f(f^\beta)) \subseteq \mathsf{fix}(f) & \\
\equiv & & \{\ \text{def. iterate}\ \} \\
& f^\alpha \subseteq \mathsf{fix}(f) &
\end{array}
$$

□

C.1.2 Proof of (4.10): $\mathcal{F}(r)^\alpha \mapsto r \in \mathcal{L}_\mathcal{E} \Rightarrow \mathcal{F}(r)^{\alpha+1} \mapsto r \in \mathcal{L}_\mathcal{E}$

Successor ordinal

1.	$\mathcal{F}(r)^\alpha \mapsto r \in L$	; ind. hyp.
2.	$W(\mathcal{F}(r)^\alpha) \mapsto \mathcal{F}(r)^\alpha \in L$	; from hyp. (c) th. 2
3.	$r \mapsto r \in L$	; hyp. (b) th. 2, SBR
4.	$r \cup W(\mathcal{F}(r)^\alpha) \mapsto r \in L$	; 3, 2 and SDR
6.	$\mathcal{F}(r)(\mathcal{F}(r)^\alpha) \mapsto r \in L$	; 5 and (4.4)
7.	$\mathcal{F}(r)^{\alpha+1} \mapsto r \in L$	; 6 and def. iterate

□

C.2 Proofs Concerning Weak Fairness

C.2.1 Termination Set of Fair Loop: $\mathbf{pre}(Y(q)(G)) = \mathbf{fix}(\overline{q} \cap G(\varnothing) \Longrightarrow (\overline{S(q)} \mid S))$

$$
\begin{array}{ll}
& \mathsf{pre}(Y(q)(G)) \\
= & \qquad \{ \text{ def. of } \mathsf{pre} \} \\
& Y(q)(G)(u) \\
= & \qquad \{ \text{ (4.16) } \} \\
& q \cup ((S \mathbin{;} Y(q)(G)) \bigtriangledown (\mathsf{grd}(G) \mid G))(u) \\
= & \qquad \{ \text{ (4.2), } X = (S \mathbin{;} Y(q)(G)),\ Z = (\mathsf{grd}(G) \mid G) \ \} \\
& q \cup ((X(u) \cup Z(u)) \cap (\overline{X(\varnothing)} \cup Z(u)) \cap (X(u) \cup \overline{Z(\varnothing)})) \\
= & \qquad \{ \ G(u) = u,\ Z(u) = \mathsf{grd}(G),\ \overline{Z(\varnothing)} = u \ \} \\
& q \cup ((X(u) \cup \mathsf{grd}(G)) \cap (\overline{X(\varnothing)} \cup \mathsf{grd}(G))) \\
= & \qquad \{ \text{ distributivity } \} \\
& q \cup \mathsf{grd}(G) \cup (X(u) \cap \overline{X(\varnothing)}) \\
= & \qquad \{ \ X = (S \mathbin{;} Y(q)(G)), \text{ set transformer } \} \\
& q \cup \mathsf{grd}(G) \cup (S(Y(q)(G)(u)) \cap \overline{S(Y(q)(G)(\varnothing))}) \\
= & \qquad \{ \text{ def. } \mathsf{grd}(Y(q)(G)), \mathsf{grd}(G) \text{ and set transformer } \} \\
& (\overline{q} \cap G(\varnothing) \Longrightarrow (\overline{S(q)} \mid S))(Y(q)(G)(u)) \\
= & \qquad \{ \text{ extreme solution of recursive equation } \} \\
& \mathsf{fix}(\overline{q} \cap G(\varnothing) \Longrightarrow (\overline{S(q)} \mid S))
\end{array}
$$

□

C.2.2 Liberal of $Y(q)(G)$: $\mathcal{L}(Y(q)(G))(r) = \mathbf{FIX}(\overline{q} \Longrightarrow (\mathbf{grd}(G) \cap G(r) \mid S))$

For $r \subseteq u \wedge r \neq u$:

$$
\begin{array}{ll}
& \mathcal{L}(Y(q)(G))(r) \\
= & \qquad \{ \text{ (4.16), Liberal set transformer of guard and dovetail } \} \\
& q \cup \mathcal{L}(S \mathbin{;} Y(q))(r) \cap \mathcal{L}(\mathsf{grd}(G) \mid G)(r) \\
= & \qquad \{ \ r \neq u,\ \mathcal{L}(\mathsf{grd}(G) \mid G)(r) = \mathsf{grd}(G) \cap \mathcal{L}(G)(r), \mathcal{L}(G)(r) = G(r) \ \} \\
& q \cup \mathcal{L}(S \mathbin{;} Y(q))(r) \cap \mathsf{grd}(G) \cap G(r) \\
= & \qquad \{ \text{ Liberal set transformer of sequencing, } \mathcal{L}(S)(r) = S(r) \ \} \\
& q \cup S(\mathcal{L}(Y(q))(r)) \cap \mathsf{grd}(G) \cap G(r) \\
= & \qquad \{ \text{ Liberal set transformer of guarded and preconditioned events } \} \\
& (\overline{q} \Longrightarrow \mathsf{grd}(G) \cap G(r) \mid S)(\mathcal{L}(Y(q)(G))(r)) \\
= & \qquad \{ \text{ extreme solution of recursive equation } \} \\
& \mathsf{FIX}(\overline{q} \Longrightarrow \mathsf{grd}(G) \cap G(r) \mid S)
\end{array}
$$

For $r = u$: $\mathcal{L}(X(q)(G))(u) = u$ First, we note that equality $\mathcal{L}(Y(q)(G))(u) = \mathsf{FIX}(\overline{q} \Longrightarrow S)$

holds:

$$
\begin{array}{ll}
& \mathcal{L}(Y(q)(G))(u) \\
= & \qquad \{\ (4.16), \text{Liberal set transformer of guard and dovetail}\ \} \\
& q \cup \mathcal{L}(S\ ;\ Y(q))(u) \cap \mathcal{L}(\mathsf{grd}(G) \mid G)(u) \\
= & \qquad \{\ \mathcal{L}(\mathsf{grd}(G) \mid G)(u) = u\ \} \\
& q \cup \mathcal{L}(S\ ;\ Y(q))(u) \\
= & \qquad \{\ S(u) = u, \text{set transformers}\ \} \\
& (\overline{q} \Longrightarrow S)(\mathcal{L}(Y(q)(G))(u)) \\
= & \qquad \{\ \text{extreme solution}\ \} \\
& \mathsf{FIX}(\overline{q} \Longrightarrow S)
\end{array}
$$

As $(\overline{q} \Longrightarrow S)(u) = u$ holds, it follows: $u \subseteq \mathsf{FIX}(\overline{q} \Longrightarrow S)$. Therefore, $\mathcal{L}(Y(q)(G))(u) = u$ follows from $u \subseteq \mathsf{FIX}(\overline{q} \Longrightarrow S)$ and equality.

C.2.3 Proof of (4.19): $\mathcal{L}(Y(q)(G))(r) \subseteq \mathbf{pre}(Y(q)(G))$

$$
\begin{array}{ll}
& \mathcal{L}(Y(q)(G))(r) \\
= & \qquad \{\ (4.18) \text{ and fixpoint property}\ \} \\
& q \cup \mathsf{grd}(G) \cap G(r) \cap S(\mathcal{L}(Y(q)(G))(r)) \\
\subseteq & \qquad \{\ \text{set theory}\ \} \\
& q \cup \mathsf{grd}(G) \\
\subseteq & \qquad \{\ \text{set theory}\ \} \\
& q \cup \mathsf{grd}(G) \cup \overline{S(q)} \cap S(\mathsf{pre}(Y(q)(G))) \\
= & \qquad \{\ \text{set transformers}\ \} \\
& (\overline{q} \cap G(\varnothing) \Longrightarrow (\overline{S(q)} \mid S))(\mathsf{pre}(Y(q)(G))) \\
= & \qquad \{\ (4.17) \text{ and fixpoint property}\ \} \\
& \mathsf{pre}(Y(q)(G))
\end{array}
$$

□

C.2.4 Monotonicity of Fair Loop: $a \subseteq b \Rightarrow Y(q)(G)(a) \subseteq Y(q)(G)(b))$

Let a and b be two subsets of u, $F(q)(a)$ and $F(q)(b)$ be the following set transformers:

$$
\begin{array}{l}
F(q)(a) = (\overline{q} \Longrightarrow G(a) \cap \mathsf{grd}(G) \mid S) \\
F(q)(b) = (\overline{q} \Longrightarrow G(b) \cap \mathsf{grd}(G) \mid S)
\end{array}
$$

$$
\begin{array}{ll}
& a \subseteq b \\
\Rightarrow & \{\ \text{Monotonicity of } G\ \} \\
& G(a) \subseteq G(b) \\
\Rightarrow & \{\ \text{set theory}\ \} \\
& \forall r \cdot (r \subseteq u \Rightarrow (q \cup G(a) \cap \mathsf{grd}(G) \cap S(r)) \subseteq (q \cup G(b) \cap \mathsf{grd}(G) \cap S(r))) \\
\equiv & \{\ \text{set transformers}\ \} \\
& \forall r \cdot (r \subseteq u \Rightarrow (\overline{q} \Longrightarrow G(a) \cap \mathsf{grd}(G) \mid S)(r) \subseteq (\overline{q} \Longrightarrow G(b) \cap \mathsf{grd}(G) \mid S)(r)) \\
\equiv & \{\ \text{def. } F(q)(a) \text{ and } F(q)(b)\ \} \\
& \forall r \cdot (r \subseteq u \Rightarrow F(q)(a)(r) \subseteq F(q)(b)(r)) \\
\Rightarrow & \{\ \text{set theory}\ \} \\
& \forall r \cdot (r \subseteq u \Rightarrow (r \subseteq F(q)(a)(r)) \Rightarrow r \subseteq F(q)(b)(r)) \\
\Rightarrow & \{\ \text{set theory}\ \} \\
& \{\, r \mid r \subseteq u \wedge r \subseteq F(q)(a)(r) \,\} \subseteq \{\, r \mid r \subseteq u \wedge r \subseteq F(q)(b)(r) \,\} \\
\Rightarrow & \{\ \text{set theory}\ \} \\
& \bigcup(\{\, r \mid r \subseteq u \wedge r \subseteq F(q)(a)(r) \,\}) \subseteq \bigcup(\{\, r \mid r \subseteq u \wedge r \subseteq F(q)(b)(r) \,\}) \\
\Rightarrow & \{\ \text{def. } \mathsf{FIX}(f),\ F(q)(a) \text{ and } F(q)(b)\ \} \\
& \mathsf{FIX}(\overline{q} \Longrightarrow G(a) \cap \mathsf{grd}(G) \mid S) \subseteq \mathsf{FIX}(\overline{q} \Longrightarrow G(b) \cap \mathsf{grd}(G) \mid S) \\
\equiv & \{\ (4.20)\ \} \\
& Y(q)(G)(a) \subseteq Y(q)(G)(b)
\end{array}
$$

□

C.2.5 Guard of Fair Loop: $\mathbf{grd}(Y(q)(G)) = \overline{q}$

$$
\begin{array}{ll}
& \mathsf{grd}(Y(q)(G)) \\
= & \{\ \text{def. guard}\ \} \\
& \overline{Y(q)(G)(\varnothing)} \\
= & \{\ (4.16)\ \} \\
& \overline{q \cup ((S\ ;\ Y(q)(G)) \triangledown (\mathsf{grd}(G) \mid G))(\varnothing)} \\
= & \{\ \text{set theory}\ \} \\
& \overline{q} \cap \overline{((S\ ;\ Y(q)(G)) \triangledown (\mathsf{grd}(G) \mid G))(\varnothing)} \\
= & \{\ \text{def. guard dovetail}\ \} \\
& \overline{q} \cap (\mathsf{grd}(S\ ;\ Y(q)(G)) \cup \mathsf{grd}(\mathsf{grd}(G) \mid G)) \\
= & \{\ \mathsf{grd}(\mathsf{grd}(G) \mid G) = u\ \} \\
& \overline{q}
\end{array}
$$

□

C.2.6 Strictness of W_w: $W_w(\varnothing) = \varnothing$

$$
\begin{array}{ll}
& W_w(\varnothing) \\
= & \qquad \{\ (4.21)\ \} \\
& \bigcup G \cdot (G \in E \mid Y(\varnothing)(G)(\varnothing)) \\
= & \qquad \{\ \text{def. of } \mathsf{grd}\ \} \\
& \bigcup G \cdot (G \in E \mid \overline{\mathsf{grd}(Y(\varnothing)(G))}) \\
= & \qquad \{\ \mathsf{grd}(Y(\varnothing)(G)) = \overline{\varnothing}\ \} \\
& \bigcup G \cdot (G \in E \mid \overline{\overline{\varnothing}}) \\
= & \\
& \varnothing
\end{array}
$$

□

C.2.7 Monotonicity of W_w: $a \subseteq b \Rightarrow W_w(a) \subseteq W_w(b)$

First we prove, for any subset a and b of u, $a \subseteq b \Rightarrow Y(a)(G)(b) \subseteq Y(b)(G)(b)$.
Let $T(a) = \mathsf{FIX}(\overline{a} \Longrightarrow (\mathsf{grd}(G) \wedge G(b) \mid S))$:

$$
\begin{array}{ll}
& a \subseteq b \\
\Rightarrow & \qquad \{\ \text{for any } G \in E\ \} \\
& a \cup \mathsf{grd}(G) \cap G(b) \cap S(T(a)) \subseteq b \cup \mathsf{grd}(G) \cap G(b) \cap S(T(a)) \\
\equiv & \qquad \{\ \text{prop. } \mathsf{FIX}\ \} \\
& T(a) \subseteq b \cup \mathsf{grd}(G) \cap G(b) \cap S(T(a)) \\
\equiv & \qquad \{\ \text{guarded set transformer}\ \} \\
& T(a) \subseteq (\overline{b} \Longrightarrow \mathsf{grd}(G) \cap G(b) \mid S)(T(a)) \\
\Rightarrow & \qquad \{\ \text{prop. } \mathsf{FIX}\ \} \\
& T(a) \subseteq \mathsf{FIX}(\overline{b} \Longrightarrow \mathsf{grd}(G) \cap G(b) \mid S) \\
\equiv & \qquad \{\ (4.20)\ \} \\
& Y(a)(G)(b) \subseteq Y(b)(G)(b)
\end{array}
$$

Now, the proof of monotonicity is:

$$
\begin{array}{ll}
& a \subseteq b \\
\Rightarrow & \qquad \{\ \text{monotonicity of } Y(q)(G) \text{ for } q = a \text{ and } G \in E\ \} \\
& Y(a)(G)(a) \subseteq Y(a)(G)(b) \\
\Rightarrow & \qquad \{\ Y(a)(G)(b) \subseteq Y(b)(G)(b)\ \} \\
& Y(a)(G)(a) \subseteq Y(b)(G)(b) \\
\Rightarrow & \\
& Y(a)(G)(a) \subseteq \bigcup G' \cdot (G' \in E \mid Y(b)(G')(b)) \\
\Rightarrow & \\
& \bigcup G \cdot (G \in E \mid Y(a)(G)(a)) \subseteq \bigcup G' \cdot (G' \in E \mid Y(b)(G')(b)) \\
\equiv & \qquad \{\ (4.21)\ \} \\
& W_w(a) \subseteq W_w(b)
\end{array}
$$

□

C.2.8 Proof of (4.26): $\forall G \cdot (G \in E \land a \mapsto b \in \mathcal{E}(G) \Rightarrow a \subseteq Y(b)(G)(b))$

1. $a \mapsto b \in \mathcal{E}(G)$; premise
2. $a \cap \overline{b} \subseteq \mathsf{grd}(G) \cap G(b) \cap S(a \cup b)$; 1 and (4.24)
3. $a \subseteq b \cup \mathsf{grd}(G) \cap G(b) \cap S(a \cup b)$; 2
4. $b \subseteq b$; trivial
5. $a \cup b \subseteq b \cup \mathsf{grd}(G) \cap G(b) \cap S(a \cup b)$; 4 and 3
6. $a \cup b \subseteq (\overline{b} \Longrightarrow \mathsf{grd}(G) \cap G(b) \mid S)(a \cup b)$; 5 and set. transf.
7. $a \cup b \subseteq \mathsf{FIX}(\overline{b} \Longrightarrow \mathsf{grd}(G) \cap G(b) \mid S)$; 6 and prop.FIX
8. $a \cup b \subseteq Y(b)(G)(b)$; 7 and (4.20)

□

C.2.9 Proof of (4.27): $\forall (G, r) \cdot (G \in E \land r \subseteq u \Rightarrow Y(r)(G)(r) \mapsto r \in \mathcal{E}(G))$

1. $S(Y(r)(G)(r)) \subseteq S(Y(r)(G)(r) \cup r)$; monotony of S
2. $Y(r)(G)(r) = r \cup \mathsf{grd}(G) \cap G(r) \cap S(Y(r)(G)(r))$; (4.20)
3. $Y(r)(G)(r) \cap \overline{r} \subseteq \mathsf{grd}(G) \cap G(r) \cap S(Y(r)(G)(r) \cup r)$; 2, 1
4. $Y(r)(G)(r) \mapsto r \in \mathcal{E}(G)$; 3 and (4.24)

□

C.3 Proofs Concerning the Derivation of Liveness Properties

C.3.1 Proof of (4.30): $\forall n \cdot (n \in \mathbb{N} \Rightarrow v'(n) = \bigcup i \cdot (i \in \mathbb{N} \land i < n \mid v(i)))$

The proof is by induction over $\mathbb{N}$. The base case:

$$
\begin{array}{ll}
& v'(0) \\
= & \qquad \{\ \text{def. } v'\ \} \\
& \{\, z \mid z \in u \land V(z) < 0 \,\} \\
= & \qquad \{\ V \in u \to \mathbb{N}\ \} \\
& \varnothing \\
= & \qquad \{\ \text{empty range}\ \} \\
& \bigcup i \cdot (i \in \mathbb{N} \land i < 0 \mid v(i))
\end{array}
$$

Inductive step:

$$
\begin{array}{ll}
& v'(n) = \bigcup i \cdot (i \in \mathbb{N} \land i < n \mid v(i)) \\
\Rightarrow & \\
& v'(n) \cup v(n) = \bigcup i \cdot (i \in \mathbb{N} \land i < n \mid v(i)) \cup v(n) \\
\Rightarrow & \qquad \{\ \text{def. } v \text{ and } v'\ \} \\
& v'(n+1) = \bigcup i \cdot (i \in \mathbb{N} \land i < n+1 \mid v(i))
\end{array}
$$

□

C.3.2 Proof of (4.31): $\bigcup i \cdot (i \in \mathbb{N} \mid v'(i+1)) = \bigcup i \cdot (i \in \mathbb{N} \mid v(i))$

$$
\begin{array}{ll}
& \bigcup i \cdot (i \in \mathbb{N} \mid v'(i+1)) \\
= & \qquad \{ \text{ def. } v' \text{ } \} \\
& \bigcup i \cdot (i \in \mathbb{N} \mid \{\, z \mid z \in u \wedge V(z) < i+1 \,\}) \\
= & \\
& \bigcup i \cdot (i \in \mathbb{N} \mid \{\, z \mid z \in u \wedge V(z) \leq i \,\}) \\
= & \\
& \bigcup i \cdot (i \in \mathbb{N} \mid \{\, z \mid z \in u \wedge V(z) = i \,\}) \\
= & \qquad \{ \text{ def. } v \text{ } \} \\
& \bigcup i \cdot (i \in \mathbb{N} \mid v(i))
\end{array}
$$

□

C.3.3 Proof of (4.32): $\bigcup i \cdot (i \in \mathbb{N} \mid v(i)) = u$

$$
\begin{array}{ll}
& u \subseteq \bigcup i \cdot (i \in \mathbb{N} \mid v(i)) \\
\Leftarrow & \qquad \{ \text{ set. theory } \} \\
& \forall x \cdot (x \in u \Rightarrow x \in \bigcup i \cdot (i \in \mathbb{N} \mid v(i))) \\
\equiv & \\
& \forall x \cdot (x \in u \Rightarrow \exists i \cdot (i \in \mathbb{N} \wedge x \in v(i))) \\
\equiv & \qquad \{ \text{ def. } v \text{ } \} \\
& \forall x \cdot (x \in u \Rightarrow \exists i \cdot (i \in \mathbb{N} \wedge V(x) = i)) \\
\Leftarrow & \\
& V \in u \rightarrow \mathbb{N}
\end{array}
$$

□

C.3.4 Proof of (4.33): $\forall n \cdot (n \in \mathbb{N} \Rightarrow \bigcup i \cdot (i \in \mathbb{N} \wedge i \leq n \mid v(i)) \cap p \subseteq f^{n+1})$

The proof is by induction. Base case:

1.	$\forall n \cdot (n \in \mathbb{N} \Rightarrow v(n) \cap p \subseteq f(v'(n)))$	; premise
2.	$v(0) \cap p \subseteq f(v'(0))$	; 1
3.	$v(0) \cap p \subseteq f(\varnothing)$	; 2 and def v'
4.	$v(0) \cap p \subseteq f^1$	; $f(\varnothing) = f^1$ (4.5)
5.	$\bigcup i \cdot (i \in \mathbb{N} \wedge i \leq 0 \mid v(i)) \cap p \subseteq f^{0+1}$	; 4

Inductive step:

1. $\bigcup i \cdot (i \in \mathbb{N} \wedge i \leq n \mid v(i)) \cap p \subseteq f^{n+1}$; Inductive hyp.
2. $f(\bigcup i \cdot (i \in \mathbb{N} \wedge i \leq n \mid v(i)) \cap p) \subseteq f(f^{n+1})$; 1 and monotonic f
3. $\forall n \cdot (n \in \mathbb{N} \Rightarrow v(n) \cap p \subseteq f(v'(n)))$; premise
4. $v(n+1) \cap p \subseteq f(v'(n+1))$; 3
5. $v'(n+1) = \bigcup i \cdot (i \in \mathbb{N} \wedge i \leq n \mid v(i))$; from (4.30)
6. $p \subseteq f(p)$; premise
7. $v(n+1) \cap p \subseteq f(v'(n+1)) \cap f(p)$; 6 and 4
8. $v(n+1) \cap p \subseteq f(v'(n+1) \cap p)$; 7 and conjunct. f
9. $v(n+1) \cap p \subseteq f(f^{n+1})$; 8, 5 and 2
10. $f^{n+1} \subseteq f^{n+2}$; from (4.5)
11. $\bigcup i \cdot (i \in \mathbb{N} \wedge i \leq n \mid v(i)) \cap p \subseteq f^{n+2}$; 10 and 1
12. $\bigcup i \cdot (i \in \mathbb{N} \wedge i \leq n \mid v(i)) \cap p \cup v(n+1) \cap p \subseteq f^{n+2}$; 11, 9 and (4.5)
13. $\bigcup i \cdot (i \in \mathbb{N} \wedge i \leq n+1 \mid v(i)) \cap p \subseteq f^{n+2}$; 12

□

C.3.5 Proof of (4.34): $\forall \alpha \cdot (\mathcal{F}_w(b)^\alpha \subseteq b \cup (\mathbf{grd}(S) \cap S(\mathbf{fix}(\mathcal{F}_w(b)))))$

The proof is given by transfinite induction considering the following abbreviations:

$$B = \mathsf{fix}(\mathcal{F}_w(b)) \tag{C.1}$$

$$F = \mathcal{F}_w(b) \tag{C.2}$$

Successor ordinal:

1. $\mathcal{F}_w(b)^\alpha \subseteq b \cup (\mathsf{grd}(S) \cap S(B))$; Ind. Hyp.
2. $\mathcal{F}_w(b)(F^\alpha) = b \cup W_w(F^\alpha)$; (4.22)
3. $\mathcal{F}_w(b)(F^\alpha) = b \cup \bigcup G \cdot (G \in E \mid Y(F^\alpha)(G)(F^\alpha))$; 2 and (4.21)
4. $\mathcal{F}_w(b)(F^\alpha) = b \cup \bigcup G \cdot (G \in E \mid \mathsf{FIX}(\overline{F^\alpha} \Longrightarrow \mathsf{grd}(G) \cap G(F^\alpha) \mid S))$; 3 and (4.20)
5. $\mathcal{F}_w(b)(F^\alpha) = b \cup \bigcup G \cdot (G \in E \mid F^\alpha \cup (\mathsf{grd}(G) \cap G(F^\alpha) \cap S(Y(F^\alpha)(G)(F^\alpha))))$; 4
6. $\mathcal{F}_w(b)(F^\alpha) \subseteq b \cup \bigcup G \cdot (G \in E \mid F^\alpha \cup (\mathsf{grd}(G) \cap S(Y(F^\alpha)(G)(F^\alpha))))$; 5
7. $\mathcal{F}_w(b)(F^\alpha) \subseteq b \cup \bigcup G \cdot (G \in E \mid F^\alpha \cup (\mathsf{grd}(G) \cap S(W_w(F^\alpha))))$; 6 and (4.21)
8. $\mathcal{F}_w(b)(F^\alpha) \subseteq b \cup \bigcup G \cdot (G \in E \mid F^\alpha \cup (\mathsf{grd}(S) \cap S(W_w(F^\alpha))))$; 7 $\mathsf{grd}(G) \subseteq \mathsf{grd}(S)$
9. $\mathcal{F}_w(b)(F^\alpha) \subseteq b \cup \bigcup G \cdot (G \in E \mid F^\alpha \cup (\mathsf{grd}(S) \cap S(\mathcal{F}_w(b)(F^\alpha))))$; 8 and (4.22)
10. $\mathcal{F}_w(b)^{\alpha+1} \subseteq b \cup F^\alpha \cup (\mathsf{grd}(S) \cap S(\mathcal{F}_w(b)^{\alpha+1}))$; 9 and (4.5)
11. $\mathcal{F}_w(b)^{\alpha+1} \subseteq b \cup F^\alpha \cup (\mathsf{grd}(S) \cap S(B))$; 10 and (4.5)
12. $\mathcal{F}_w(b)^{\alpha+1} \subseteq b \cup (\mathsf{grd}(S) \cap S(B))$; 11 and 1

Limit ordinal:

1. $\forall \beta \cdot (\beta < \alpha \Rightarrow \mathsf{F}^\beta \subseteq b \cup \mathsf{grd}(S) \cap S(B))$; Ind. Hyp
2. $\mathsf{F}(\mathsf{F}^\beta) = b \cup W_w(\mathsf{F}^\beta)$; (4.22), $\beta < \alpha$
3. $\mathsf{F}(\mathsf{F}^\beta) = b \cup \bigcup G \cdot (G \in E \mid Y(\mathsf{F}^\beta)(G)(\mathsf{F}^\beta))$; (4.21)
4. $\mathsf{F}(\mathsf{F}^\beta) = b \cup \bigcup G \cdot (G \in E \mid \mathsf{FIX}(\overline{\mathsf{F}^\beta} \Longrightarrow \mathsf{grd}(G) \cap G(\mathsf{F}^\beta) \mid S))$; 3, (4.20)
5. $\mathsf{F}(\mathsf{F}^\beta) = b \cup \bigcup G \cdot (G \in E \mid \mathsf{F}^\beta \cup (\mathsf{grd}(G) \cap G(\mathsf{F}^\beta) \cap S(Y(\mathsf{F}^\beta)(G)(\mathsf{F}^\beta))))$; 4
6. $\mathsf{F}(\mathsf{F}^\beta) \subseteq b \cup \mathsf{F}^\beta \cup (\mathsf{grd}(S) \cap S(\mathsf{F}(\mathsf{F}^\beta)))$; 5, (4.21) and (4.22)
7. $\mathsf{F}(\mathsf{F}^\beta) \subseteq b \cup \mathsf{F}^\beta \cup (\mathsf{grd}(S) \cap S(B))$; 6 and (4.8)
8. $\mathsf{F}(\mathsf{F}^\beta) \subseteq b \cup (\mathsf{grd}(S) \cap S(B))$; 7, 1 $\beta < \alpha$
9. $\bigcup \beta \cdot (b < \alpha \mid \mathsf{F}(\mathsf{F}^\beta)) \subseteq b \cup (\mathsf{grd}(S) \cap S(B))$; 8
10. $\mathsf{F}^\alpha \subseteq b \cup (\mathsf{grd}(S) \cap S(B))$; 9 and (4.5)

Using (4.34), the proof of sufficient conditions is as follows:

1.	$\forall n \cdot (n \in \mathbb{N} \Rightarrow \overline{b} \cap v(n) \subseteq S(v'(n)))$	; premise
2.	$a \mapsto b \in \mathcal{L}_{\mathcal{E}_w}$	; premise
3.	$\forall \alpha' \cdot (\mathcal{F}_w(b)^{\alpha'} \subseteq b \cup (\mathsf{grd}(S) \cap S(\mathsf{fix}(\mathcal{F}_w(b)))))$	; (4.34)
4.	$\exists \alpha \cdot (\mathsf{fix}(\mathcal{F}_w(b)) = \mathcal{F}_w(b)^{\alpha})$	; Theorem 1
5.	$\mathsf{fix}(\mathcal{F}_w(b)) \subseteq b \cup (\mathsf{grd}(S) \cap S(\mathsf{fix}(\mathcal{F}_w(b))))$	; 4 and 3
6.	$\mathsf{fix}(\mathcal{F}_w(b)) \subseteq \mathcal{F}_m(b)(\mathsf{fix}(\mathcal{F}_w(b)))$	; 5 and (4.12)
7.	$a \mapsto b \in \mathcal{T}_w$	; 2, equality (4.28)
8.	$a \subseteq \mathsf{fix}(\mathcal{F}_w(b))$	; 7 and def. $\mathcal{T}_w$
9.	$\mathsf{fix}(\mathcal{F}_w(b)) \cap \overline{b} \subseteq \mathsf{grd}(S)$	; 5
10.	$\forall n \cdot (n \in \mathbb{N} \Rightarrow \overline{b} \cap \mathsf{fix}(\mathcal{F}_w(b)) \cap v(n) \subseteq \mathsf{grd}(S) \cap S(v'(n)))$	; 9 and 1
11.	$\forall n \cdot (n \in \mathbb{N} \Rightarrow \mathsf{fix}(\mathcal{F}_w(b)) \cap v(n) \subseteq b \cup \mathsf{grd}(S) \cap S(v'(n)))$	; 10
12.	$\forall n \cdot (n \in \mathbb{N} \Rightarrow \mathsf{fix}(\mathcal{F}_w(b)) \cap v(n) \subseteq \mathcal{F}_m(b)(v'(n)))$	; 11 and (4.12)
13.	$\mathsf{fix}(\mathcal{F}_w(b)) \subseteq \mathsf{fix}(\mathcal{F}_m(b))$	; 12, 6 and th. 3
14.	$a \subseteq \mathsf{fix}(\mathcal{F}_m(b))$	; 13 and 8
15.	$a \mapsto b \in \mathcal{T}_m$	; 14 and def. $\mathcal{T}_m$
16.	$a \mapsto b \in \mathcal{L}_{\mathcal{E}_m}$	; 15, equality (4.15)

□

Appendix D

Proofs of Chapter 5

D.1 Proofs Concerning Minimal Progress

D.1.1 Proof of the Equivalence between BMP and (5.4)

$$
\begin{array}{ll}
& \forall n \cdot (n \in W \Rightarrow w(n) \subseteq H(w'(n))) \\
\equiv & \\
& \forall (n, y) \cdot (n \in W \wedge y \in w(n) \Rightarrow y \in H(w'(n))) \\
\equiv & \{\ \text{def. } w(n)\ \} \\
& \forall (n, y) \cdot (n \in W \wedge y \in v \wedge V(y) = n \Rightarrow y \in H(w'(n))) \\
\equiv & \{\ \text{def. of set transformer and } w'\ \} \\
& \forall (n, y) \cdot (n \in W \wedge y \in v \wedge V(y) = n \Rightarrow [H]\, y \in v \wedge V(y) < n) \\
\equiv & \{\ \text{def. of } v \text{ and } y \in v \Rightarrow [H]\, y \in v\ \} \\
& \forall (n, y) \cdot (n \in W \wedge I \wedge J \wedge V(y) = n \Rightarrow [H]\, V(y) < n)
\end{array}
$$

□

D.1.2 Proof of the Equivalence between LMP and (5.5)

$$
\begin{array}{ll}
& r^{-1}[\overline{S(\varnothing)}] \subseteq \overline{(T \,[\!]\, H)(\varnothing)} \\
\equiv & \\
& \forall y \cdot (y \in r^{-1}[\overline{S(\varnothing)}] \Rightarrow y \in \overline{(T \,[\!]\, H)(\varnothing)}) \\
\equiv & \{\ \text{def. set transformer}\ \} \\
& \forall y \cdot (y \in r^{-1}[\overline{S(\varnothing)}] \Rightarrow y \in \overline{\{\, z \mid z \in v \wedge [T \,[\!]\, H]\, z \in \varnothing \,\}}) \\
\equiv & \{\ \text{fact of sets}\ \} \\
& \forall y \cdot (y \in r^{-1}[\overline{S(\varnothing)}] \Rightarrow y \in \{\, z \mid z \in v \wedge \neg [T \,[\!]\, H]\, \mathit{false} \,\}) \\
\equiv & \{\ \text{def. of } \mathit{grd}\ \} \\
& \forall y \cdot (y \in r^{-1}[\overline{S(\varnothing)}] \Rightarrow y \in v \wedge (\mathit{grd}(T) \vee \mathit{grd}(H))) \\
\equiv & \{\ y \in r^{-1}[\overline{S(\varnothing)}] \Rightarrow y \in v\ \} \\
& \forall y \cdot (y \in r^{-1}[\overline{S(\varnothing)}] \Rightarrow \mathit{grd}(T) \vee \mathit{grd}(H)) \\
\equiv & \{\ x \in \overline{S(\varnothing)} \Rightarrow x \in u \wedge \mathit{grd}(S)\ \} \\
& \forall y \cdot (\exists x \cdot (x \in u \wedge \mathit{grd}(S) \wedge y \mapsto x \in r) \Rightarrow \mathit{grd}(T) \vee \mathit{grd}(H)) \\
\equiv & \{\ \text{def } r \text{ and pred. calc.}\ \} \\
& \forall (y, x) \cdot (I \wedge J \wedge \mathit{grd}(S) \Rightarrow \mathit{grd}(T) \vee \mathit{grd}(H))
\end{array}
$$

□

D.1.3 Proof of (5.9): $p' \cap \overline{q'} \subseteq r^{-1}[p \cap \overline{q}]$

1.	$y \in p' \cap \overline{q'}$	; premise
2.	$y \in p' \wedge \neg(y \in q')$	; 1
3.	$y \in r^{-1}[p] \wedge \neg(y \in r^{-1}[q])$	; 2 and def p' and q'
4.	$\exists x \cdot (x \in p \wedge x \mapsto y \in r^{-1}) \wedge \neg(\exists x \cdot (x \in q \wedge x \mapsto y \in r^{-1}))$	; 3
5.	$\exists x \cdot (x \in p \wedge x \mapsto y \in r^{-1}) \wedge \forall x \cdot (x \mapsto y \in r^{-1} \Rightarrow x \notin q)$	; 4
6.	$\exists x \cdot (x \in p \wedge x \notin q \wedge x \mapsto y \in r^{-1})$	; 5
7.	$y \in r^{-1}[p \cap \overline{q}]$	; 6

□

D.2 Proofs Concerning Weak Fairness

D.2.1 Proof of the PSP Theorem

PSP Theorem under Weak Fairness:

$$\frac{p \mapsto q \in \mathcal{L}_{\mathcal{E}_w}, a \cap \overline{b} \subseteq S(a \cup b)}{p \cap a \mapsto q \cap a \cup b \in \mathcal{L}_{\mathcal{E}_w}}$$

The proof is given by structural induction in the definition of $\mathcal{L}_{\mathcal{E}_w}$. Let $P(p, q, a, b)$ be a short hand for the conclusion of the PSP theorem:

$$P(p, q, a, b) \equiv p \cap a \mapsto q \cap a \cup b \in \mathcal{L}_{\mathcal{E}_w}$$

Three proofs are required:

Base If $p \mapsto q \in \mathcal{E}_w$ and $a \cap \overline{b} \subseteq S(a \cup b)$ holds then $P(p, q, a, b)$ follows.

Transitivity If $a \cap \overline{b} \subseteq S(a \cup b)$, $\{p \mapsto o, o \mapsto q\} \subseteq \mathcal{L}_{\mathcal{E}_w}$, $P(p, o, a, b)$ and $P(o, q, a, b)$ holds then $P(p, q, a, b)$ follows.

Disjunction If $a \cap \overline{b} \subseteq S(a \cup b)$, $l \times \{q\} \subseteq \mathcal{L}_{\mathcal{E}_w}$ and $\forall p \cdot (p \in l \Rightarrow P(p, q, a, b))$ holds then $P(\bigcup(l), q, a, b)$ follows.

Proof of the Base Case

From $p \mapsto q \in \mathcal{E}_w$ and definition of $\mathcal{E}_w$ follows:

$$p \cap \overline{q} \subseteq S(p \cup q) \cap G(q) \cap \mathsf{grd}(G)$$

for some helpful event G. Taking the conjunction of this derivation with the premise $a \cap \overline{b} \subseteq S(a \cup b)$ allows the derivation of:

$$p \cap \overline{q} \cap a \cap \overline{b} \subseteq S(p \cap a \cup q \cap a \cup b) \cap G(q \cap a \cup b) \cap \mathsf{grd}(G)$$

From this inclusion and the definition of $\mathcal{E}_w$ follows: $p \cap a \mapsto q \cap a \cup b \in \mathcal{E}_w$ which allows the proof of the goal $P(p, q, a, b)$.

Proof of the Transitivity Case

$b \mapsto q \cap a \cup b \in \mathcal{L}_{\mathcal{E}_w}$ holds trivially. From this derivation and the hypothesis $P(o, q, a, b)$, by application of the SDR rule of definition 4.4, follows:

$$o \cap a \cup b \mapsto q \cap a \cup b \in \mathcal{L}_{\mathcal{E}_w}$$

Finally, from this derivation and the hypothesis $P(p, o, a, b)$ follows the goal $P(p, q, a, b)$ by application of the STR rule of definition 4.4.

Proof of the Disjunction Case

The goal $P(\bigcup(l), q, a, b)$ follows directly from the hypothesis $\forall p \cdot (p \in l \Rightarrow P(p, q, a, b))$ by application of the SDR rule of definition 4.4.

□

D.2.2 Proof of (5.16): $r^{-1}[p \cap \overline{q}] \cap \mathbf{grd}(G') \subseteq (F' \llbracket H)(\mathbf{grd}(G'))$

$$
\begin{array}{ll}
& r^{-1}[p \cap \overline{q}] \cap \mathsf{grd}(G') \subseteq (F' \llbracket H)(\mathsf{grd}(G')) \\
\equiv & \\
& \forall y \cdot (y \in r^{-1}[p \cap \overline{q}] \wedge y \in \mathsf{grd}(G') \Rightarrow y \in (F' \llbracket H)(\mathsf{grd}(G'))) \\
\equiv & \{\ \text{set transformers}\ \} \\
& \forall y \cdot (y \in r^{-1}[p \cap \overline{q}] \wedge y \in v \wedge grd(G') \Rightarrow [F' \llbracket H]\,(y \in v \wedge grd(G'))) \\
\equiv & \{\ y \in v \Rightarrow [F' \llbracket H]\, y \in v\ \} \\
& \forall y \cdot (y \in r^{-1}[p \cap \overline{q}] \wedge y \in v \wedge grd(G') \Rightarrow [F' \llbracket H]\, grd(G')) \\
\equiv & \{\ \text{def. of } p \text{ and } q\ \} \\
& \forall y \cdot (y \in v \wedge \exists x \cdot (I(x) \wedge J(x, y) \wedge P(x) \wedge \neg Q(x)) \wedge grd(G') \Rightarrow [F' \llbracket H]\, grd(G')) \\
\equiv & \{\ \exists x \cdot (I \wedge J \wedge P \wedge \neg Q) \Rightarrow y \in v \text{ and pred. calc.}\ \} \\
& \forall (x, y) \cdot (I(x) \wedge J(x, y) \wedge P(x) \wedge \neg Q(x) \wedge grd(G') \Rightarrow [F' \llbracket H]\, grd(G'))
\end{array}
$$

D.2.3 Proof of (5.17): $r^{-1}[p \cap \overline{q}] \cap \overline{\mathbf{grd}(G')} \mapsto \mathbf{grd}(G') \in \mathcal{L}_{\mathcal{E}'_w}$

In order to prove the equivalence between (5.17) and LIP proof obligation, we need the following theorem about *leads to*

$$\frac{(\exists x \cdot (P(x)) \wedge Q) \rightsquigarrow R\ ,\ x \backslash Q\ ,\ x \backslash R}{(P(x) \wedge Q) \rightsquigarrow R} \tag{D.1}$$

which is proved as follows:

1.	$(\exists x \cdot (P(x)) \wedge Q) \rightsquigarrow R$	; premise
2.	$(\exists y \cdot (P(y)) \wedge Q) \rightsquigarrow R$	; 1
3.	$P(x) \Rightarrow \exists y \cdot (P(y))$	; for any x
4.	$P(x) \wedge Q \Rightarrow (\exists y \cdot (P(y)) \wedge Q)$	; 3
5.	$(P(x) \wedge Q) \rightsquigarrow (\exists y \cdot (P(y)) \wedge Q)$	; 4 and BRL
6.	$(P(x) \wedge Q) \rightsquigarrow R$	; TRA 5 and 2

Now, the equivalence between (5.17) and LIP proof obligation is as follows:

$$
\begin{aligned}
& r^{-1}[p \cap \overline{q}] \cap \overline{\mathsf{grd}(G')} \mapsto \mathsf{grd}(G') \in \mathcal{L}_{\mathcal{E}'_w} \\
\equiv & \quad \{ \text{ equivalence (4.7) instantiated to the refinement } \mathcal{T} \ \} \\
& y \in r^{-1}[p \cap \overline{q}] \cap \overline{\mathsf{grd}(G')} \rightsquigarrow y \in \mathsf{grd}(G') \\
\equiv & \quad \{ \ y \in \mathsf{grd}(G') \equiv y \in v \wedge grd(G') \text{ and } y \in v \text{ is invariant } \} \\
& y \in r^{-1}[p \cap \overline{q}] \wedge \neg grd(G') \rightsquigarrow grd(G') \\
\equiv & \quad \{ \text{ def. } r, p \text{ and } q \ \} \\
& \exists x \cdot (P(x) \wedge \neg Q(x) \wedge I(x) \wedge J(y,x)) \wedge \neg grd(G') \rightsquigarrow grd(G') \\
\equiv & \quad \{ \text{ DSJ rule, } x \backslash grd(G') \text{ and (D.1) } \} \\
& P(x) \wedge \neg Q(x) \wedge I(x) \wedge J(y,x) \wedge \neg grd(G') \rightsquigarrow grd(G')
\end{aligned}
$$

□

Appendix E

Proof of Rules R1 and R2

The proof of rules R1 and R2 needs a generalization of both the state space of a refinement, and the refinement relation. Refinements are numbered from 1 to n, where $n \in \mathbb{N}_1$. The state space and the refinement relation in refinement i are denoted by v_i and r_i, defined as comes next:

$$v_i = \{\, d' \mid \exists d \cdot (d \in v_{i-1} \wedge J_i(d', d)) \,\} \tag{E.1}$$

$$r_i = \{\, d' \mapsto d \mid d \in v_{i-1} \wedge J_i(d', d) \,\} \tag{E.2}$$

It is assumed that v_0 is the state space of the abstract model, and it is equal to u, that is, the set of states satisfying the invariant I. Moreover, in refinement i, the state variable is denoted by y_i and the gluing invariant is J_i. This invariant relates the state variable y_i with y_{i-1}; y_0 is equal to x, the state variable of the abstract model. $\mathcal{R}_i$ denotes the sequential composition of refinement relations r_i until r_1:

$$\mathcal{R}_i = (r_i \,;\, r_{i-1} \,;\, \ldots r_1) \tag{E.3}$$

E.1 Proof of Rule R1

From the premise or rule R1, $G_i \cdot P \gg Q$ follows from the WF0 and WF1 proof obligations, which are equivalent, in a set theoretical framework to:

$$p \cap \overline{q} \subseteq S_i(p \cup q) \tag{E.4}$$

$$p \cap \overline{q} \subseteq G_i(q) \cap \mathsf{grd}(G_i) \tag{E.5}$$

where S_i is the choice of events in refinement i, and p and q the subsets of v_i where P and Q hold respectively. From (E.4) and (E.5), with a calculus similar to the proofs of the inclusion 5.21 in section 5.3, the following inclusions are derived:

$$p' \cap \overline{q'} \subseteq S_{i+1}(p' \cup q') \tag{E.6}$$

$$p' \cap \overline{q'} \subseteq G_{i+1}(q') \tag{E.7}$$

where S_{i+1} and G_i are the choice of events and the helpful event in refinement $i+1$ respectively, and the sets $p' = r_{i+1}^{-1}[p]$ and $q' = r_{i+1}^{-1}[q]$. Now, from the premise of rule R1, the ensuing inclusion holds:

$$r_{i+1}^{-1}[p \cap \overline{q}] \subseteq \mathsf{grd}(G_{i+1}) \tag{E.8}$$

Finally, from (E.6), (E.7) and (E.8) follows: $G_{i+1} \cdot y_{i+1} \in p' \gg y_{i+1} \in q'$, which guarantees the preservation of the basic property in refinement $i+1$.

E.2 Proof of Rule R2 (LIP)

Using (E.3), the set theoretical counterpart of the LIP property in refinement i is:

$$y_i \in \mathcal{R}_i^{-1}[p \cap \overline{q}] \cap G_i(\varnothing) \rightsquigarrow y_i \in \overline{G_i(\varnothing)} \tag{E.9}$$

where G_i denotes the refinement of the helpful event G. According to the premise of rule R2, (E.9) follows from a set of basic liveness properties β, and any b in β is preserved in refinement $i+1$. Therefore, the property (E.9), considered as a general liveness property, is preserved in refinement $i+1$, which allows the derivation of the following property in refinement $i+1$:

$$y_{i+1} \in r_{i+1}^{-1}[\mathcal{R}_i^{-1}[p \cap \overline{q}] \cap G_i(\varnothing)] \rightsquigarrow y_{i+1} \in r_{i+1}^{-1}[\overline{G_i(\varnothing)}] \tag{E.10}$$

Now, using the following inclusions:

$$r_{i+1}^{-1}[\overline{G_i(\varnothing)}] \subseteq \overline{G_{i+1}(\varnothing)} \tag{E.11}$$

$$\mathcal{R}_{i+1}^{-1}[p \cap \overline{q}] \cap G_{i+1}(\varnothing) \subseteq r_{i+1}^{-1}[\mathcal{R}_i^{-1}[p \cap \overline{q}] \cap G_i(\varnothing)] \tag{E.12}$$

along with the rules BRL and TRA, the following property is proved:

$$y_{i+1} \in \mathcal{R}_{i+1}^{-1}[p \cap \overline{q}] \cap G_{i+1}(\varnothing) \rightsquigarrow y_{i+1} \in \overline{G_{i+1}(\varnothing)} \tag{E.13}$$

which is equivalent to LIP proof obligation in refinement $i+1$:

$$y_{i+1} \in \mathcal{R}_{i+1}^{-1}[p \cap \overline{q}] \wedge \neg grd(G_{i+1}) \rightsquigarrow grd(G_{i+1}) \tag{E.14}$$

Inclusion (E.11) is the set theoretical counterpart of the premise $I \wedge \mathcal{J}_i \wedge J_{i+1} \wedge grd(G_i) \Rightarrow grd(G_{i+1})$, and (E.12) follows from (E.11).

E.3 Proof of Rule R2 (SAP)

The calculations in this proof are similar to those which are presented in section 5.3. The set theoretical counterpart of SAP proof obligation in refinement i is:

$$\mathcal{R}_i[p \cap \overline{q}] \cap \overline{G_i(\varnothing)} \subseteq e(\overline{G_i(\varnothing)}) \tag{E.15}$$

Let E_{i+1} be the set of events in refinement $i+1$, and G_{i+1} the refinement of G_i. E_{i+1} is made up of two disjoint subsets of events: E'_{i+1}, containing the refined events, and H_{i+1}, which includes the new events in refinement $i+1$. Each refined event e' of E'_{i+1}, associated with an event e of E_i, and each new event h of H_{i+1}, satisfy the refinement relations:

$$e(\overline{r_{i+1}[\overline{s}]}) \subseteq \overline{r_{i+1}[\overline{e'(s)}]} \tag{E.16}$$

$$skip(\overline{r_{i+1}[\overline{s}]}) \subseteq \overline{r_{i+1}[\overline{h(s)}]} \tag{E.17}$$

where s is universally quantified over $\mathbb{P}(v_{i+1})$. From refinement relations follow:

$$r_{i+1}^{-1}[G_i(\varnothing)] \subseteq G_{i+1}(\varnothing) \tag{E.18}$$

$$\overline{G_{i+1}(\varnothing)} \subseteq r_{i+1}^{-1}[\overline{G_i(\varnothing)}] \tag{E.19}$$

$$\forall h \cdot (h \in H_{i+1} \Rightarrow r_{i+1}^{-1}[\overline{G_i(\varnothing)}] \subseteq h(r_{i+1}^{-1}[\overline{G_i(\varnothing)}])) \tag{E.20}$$

and from (E.15) and (E.16) follows:

$$\forall e' \cdot (e' \in E'_{i+1} - \{G_{i+1}\} \Rightarrow (r_{i+1}^{-1}[\mathcal{R}_i[p \cap \overline{q}] \cap \overline{G_i(\varnothing)}] \subseteq e'(r_{i+1}^{-1}[\overline{G_i(\varnothing)}]))) \tag{E.21}$$

Now, from (E.18), follows:

$$\mathcal{R}_{i+1}^{-1}[p \cap \overline{q}] \cap \overline{G_{i+1}(\varnothing)} \subseteq r_{i+1}^{-1}[\mathcal{R}_i^{-1}[p \cap \overline{q}] \cap \overline{G_i(\varnothing)}] \tag{E.22}$$

From (E.11), (E.21) and (E.22), follows that concrete events in refinement $i+1$ preserve the guard of G_{i+1}:

$$\forall e' \cdot (e' \in E'_{i+1} - \{G_{i+1}\} \Rightarrow \mathcal{R}_{i+1}^{-1}[p \cap \overline{q}] \cap \overline{G_{i+1}(\varnothing)} \subseteq e'(\overline{G_{i+1}(\varnothing)})) \tag{E.23}$$

New events in refinement $i+1$ do not modify the state variable in refinement i, therefore, from (E.11), (E.20) and (E.22) follows:

$$\forall h \cdot (h \in H_{i+1} \Rightarrow \mathcal{R}_{i+1}^{-1}[p \cap \overline{q}] \cap \overline{G_{i+1}(\varnothing)} \subseteq h(\overline{G_{i+1}(\varnothing)})) \tag{E.24}$$

Finally, from (E.23) and (E.24), follows the following property:

$$\forall e' \cdot (e' \in E_{i+1} - \{G_{i+1}\} \Rightarrow \mathcal{R}_{i+1}^{-1}[p \cap \overline{q}] \cap \overline{G_{i+1}(\varnothing)} \subseteq e'(\overline{G_{i+1}(\varnothing)}))$$

which is equivalent to SAP proof obligation in refinement $i+1$.

Appendix F

Verifying Semantic Proofs

In this appendix, we present some abstract machines used to verify some proofs in the set theoretical framework of Chapter 4. The general idea in these abstract machines is to define sets and set transformers as constants then, we define different properties (*axioms*) of these constants and finally, we state theorems about these constants to be proved.

We present five machines. The first machine, *dovetail*, defines the dovetail operator and it is used to prove the alternative definition of its termination set and its guard. The second machine, *fairloop*, defines the fair loop used in the section 4.2.4 and proves the recursive definition of its liberal set transformer; this proof gives as outcome the definition (4.18). Machine *propfair* defines the greatest fixpoint as a generalized union of sets and it is mainly used to prove the monotonicity of the fair loop. Machine *guardfair* is used to prove the guard of the fair loop. Finally, Machine *loopterm* is used to prove the total correctness of the fair loop (C.2.8).

Machine *dovetail*

The dovetail operator is denoted by the function Dtl. Therefore, if S and T are set transformers, the set transformer $S \triangledown T$, is denoted as $Dtl(S, T)$.

MACHINE
dovetail

SETS
uu

CONSTANTS
St, Dtl, Li, cpl

PROPERTIES

/* The set of set transformers */
$St = \{\, SS \mid SS \in \mathbb{P}(uu) \rightarrow \mathbb{P}(uu) \,\} \wedge$

/* Type of Dovetail */

$Dtl \in St \times St \to St \wedge$

/* Type of Liberal set transformer */
$Li \in St \to St \wedge$

/* Liberal set transformer for dovetail */
$\forall(SS, TT) \cdot (SS \in St \wedge TT \in St \Rightarrow \forall rr \cdot (rr \in \mathbb{P}(uu) \Rightarrow$
$Li(Dtl(SS, TT))(rr) = Li(SS)(rr) \cap Li(TT)(rr))) \wedge$

/* Complement of any set */
$cpl = \lambda rr \cdot (rr \in \mathbb{P}(uu) \mid uu - rr) \wedge$

/* Termination set for dovetail */
$\forall(SS, TT) \cdot (SS \in St \wedge TT \in St \Rightarrow Dtl(SS, TT)(uu) =$
$((SS(uu) \cup TT(uu)) \cap (cpl(SS(\varnothing)) \cup TT(uu)) \cap (cpl(TT(\varnothing)) \cup SS(uu)))) \wedge$

/* Set transformer for dovetail */
$\forall(SS, TT) \cdot (SS \in St \wedge TT \in St \Rightarrow \forall rr \cdot (rr \in \mathbb{P}(uu) \Rightarrow$
$Dtl(SS, TT)(rr) = Li(Dtl(SS, TT))(rr) \cap Dtl(SS, TT)(uu))) \wedge$

/* Pairing condition */
$\forall(SS, rr) \cdot (SS \in St \wedge rr \in \mathbb{P}(uu) \Rightarrow SS(rr) = Li(SS)(rr) \cap SS(uu))$

ASSERTIONS

/* Alternative definition of the termination set */
$\forall(SS, TT) \cdot (SS \in St \wedge TT \in St \Rightarrow Dtl(SS, TT)(uu) =$
$((SS(uu) \cap TT(uu)) \cup (cpl(SS(\varnothing)) \cap SS(uu)) \cup (cpl(TT(\varnothing)) \cap TT(uu))))$;

/* Guard */
$\forall(SS, TT) \cdot (SS \in St \wedge TT \in St \Rightarrow cpl(Dtl(SS, TT)(\varnothing)) = cpl(SS(\varnothing)) \cup cpl(TT(\varnothing)))$

END

Machine *fairloop*

In this machine the fair loop is defined. Sequential composition of set transformers S **;** T is noted as $Seq(S, T)$. A guarded set transformer $p \Longrightarrow S$ is noted as $Grd(p, S)$.

In section 4.2.4, the fair loop is noted $Y(q)(G)$. In this machine it is denoted as $XX(oo)$. The set transformer S in the definition of the fair loop (4.16) is denoted by FF in this machine. The non recursive branch $grd(G) \mid G$ in $Y(q)(G)$ is simply denoted by GG.

MACHINE
fairloop

SEES
dovetail

CONSTANTS
Seq, Grd, Mn, FF, GG, XX

PROPERTIES

/* Sequencing set transformer */
$Seq \in St \times St \to St \wedge$

/* Guarding set transformer */
$Grd \in \mathbb{P}(uu) \times St \to St \wedge$

/* Set Monotonic Set Transformers */
$Mn = \{\, SS \mid SS \in St \wedge \forall (ss, tt) \cdot ($
$ss \in \mathbb{P}(uu) \wedge tt \in \mathbb{P}(uu) \wedge ss \subseteq tt \Rightarrow SS(ss) \subseteq SS(tt)) \,\} \wedge$

/* Two monotonic set transformers */
$FF \in Mn \wedge$
$GG \in Mn \wedge$
$FF(uu) = uu \wedge$
$GG(uu) = uu \wedge$

/* The fair loop */
$XX \in \mathbb{P}(uu) \to St \wedge$
$XX = \lambda oo \cdot (oo \in \mathbb{P}(uu) \mid \lambda rr \cdot (rr \in \mathbb{P}(uu) \mid$
$Grd(cpl(oo), Dtl(Seq(FF, XX(oo)), GG))(rr))) \wedge$

/* Liberal set transformer of sequencing */
$\forall (SS, TT, rr) \cdot (SS \in St \wedge TT \in St \wedge rr \in \mathbb{P}(uu) \Rightarrow$
$Li(Seq(SS, TT))(rr) = Li(SS)(Li(TT)(rr))) \wedge$

/* Liberal set transformer of guarding */
$\forall (oo, SS, rr) \cdot (oo \in \mathbb{P}(uu) \wedge SS \in St \wedge rr \in \mathbb{P}(uu) \Rightarrow$
$Li(Grd(oo, SS))(rr) = cpl(oo) \cup Li(SS)(rr)) \wedge$

/* Liberal set transformer of lambda */
$\forall (SS, rr) \cdot (SS \in St \wedge rr \in \mathbb{P}(uu) \Rightarrow Li(\lambda ss \cdot (ss \in \mathbb{P}(uu) \mid SS(ss)))(rr) = Li(SS)(rr))$

ASSERTIONS

/* Recursive definition */
$\forall (oo, rr) \cdot (oo \in \mathbb{P}(uu) \wedge rr \in \mathbb{P}(uu) \Rightarrow$
$Li(XX(oo))(rr) = oo \cup (Li(FF)(Li(XX(oo))(rr)) \cap Li(GG)(rr)))$

END

Machine *propfair*

MACHINE
propfair

SEES
$dovetail, fairloop$

CONSTANTS
FIX, ff

PROPERTIES

/* Definition of greatest fixpoint */
$FIX = \lambda SS \cdot (SS \in Mn \mid union(\{\, xx \mid xx \in \mathbb{P}(uu) \wedge xx \subseteq SS(xx) \,\})) \wedge$

/* Definition of Liberal Set Transformer for recursive functions */
$\forall (SS, TT, rr) \cdot (SS \in St \wedge TT \in Mn \wedge rr \in \mathbb{P}(uu) \wedge Li(SS)(rr) = TT(Li(SS)(rr)) \Rightarrow$
$Li(SS)(rr) = \mathsf{FIX}(TT)) \wedge$

/* Property of Liberal Set Transformers */
$\forall (SS, rr) \cdot (SS \in Mn \wedge rr \in \mathbb{P}(uu) \wedge SS(uu) = uu \Rightarrow Li(SS)(rr) = SS(rr)) \wedge$

/* Set Transformer of the Fair Loop XX(oo) */
$\forall (rr, oo) \cdot (rr \in \mathbb{P}(uu) \wedge oo \in \mathbb{P}(uu) \Rightarrow XX(oo)(rr) = Li(XX(oo))(rr) \cap XX(oo)(uu)) \wedge$

/* Auxiliary Function */
$ff = \lambda ss \cdot (ss \in \mathbb{P}(uu) \mid \lambda tt \cdot (tt \in \mathbb{P}(uu) \mid \lambda rr \cdot (rr \in \mathbb{P}(uu) \mid ss \cup (GG(tt) \cap FF(rr)))))$

ASSERTIONS

/* For any rr in $\mathbb{P}(uu)$, $ff(oo)(rr)$ is monotone */
$\forall (rr, oo) \cdot (rr \in \mathbb{P}(uu) \wedge oo \in \mathbb{P}(uu) \Rightarrow ff(oo)(rr) \in Mn)$;

/* The Liberal Set Transformer of $XX(oo)$ */
$\forall (rr, oo) \cdot (rr \in \mathbb{P}(uu) \wedge oo \in \mathbb{P}(uu) \Rightarrow Li(XX(oo))(rr) = \mathsf{FIX}(ff(oo)(rr)))$;

/* The Liberal Set Transformer of XX(oo) is monotone */
$\forall oo \cdot (oo \in \mathbb{P}(uu) \Rightarrow Li(XX(oo)) \in Mn)$;

/* The Set Transformer of XX(oo) is monotone */
$\forall oo \cdot (oo \in \mathbb{P}(uu) \Rightarrow XX(oo) \in Mn)$
END

Machine *guardfair*

In this machine we prove that the guard of the fair loop $XX(oo)$ is the least fixpoint of the function $ff(oo)(\varnothing)$. In Section 4.2.4, we prove that the guard of the fair loop $Y(q)(G)$ is equal to $\overline{q}$. We remark that changing the set transformer GG, in the definition of $XX(oo)$, by the preconditioned set transformer $\mathsf{grd}(GG) \mid GG$, as it is done in $Y(q)(G)$, we obtain that the guard of $XX(oo)$ is equal to the complement of oo. This result follows by remarking that $XX(oo)(\varnothing) = ff(oo)(\varnothing)(\mathsf{fix}(ff(oo)(\varnothing)))$ and by replacing GG by $\mathsf{grd}(GG) \mid GG$ in the definition of ff.

MACHINE
guardfair

SEES
dovetail, *fairloop*, *propfair*

CONSTANTS
fix

PROPERTIES

/* Type of least fixpoint */
$fix \in Mn \rightarrow \mathbb{P}(uu) \wedge$

/* Definition of Set Transformer for recursive functions */
$\forall(SS, TT, rr) \cdot (SS \in St \wedge TT \in Mn \wedge rr \in \mathbb{P}(uu) \wedge SS(rr) = TT(SS(rr)) \Rightarrow SS(rr) = \mathsf{fix}(TT)) \wedge$

/* Set transformer of guarding */
$\forall(oo, SS, rr) \cdot (oo \in \mathbb{P}(uu) \wedge SS \in St \wedge rr \in \mathbb{P}(uu) \Rightarrow Grd(oo, SS)(rr) = \overline{oo} \cup SS(rr)) \wedge$

/* Set transformer of sequencing */
$\forall(SS, TT, rr) \cdot (SS \in St \wedge TT \in St \wedge rr \in \mathbb{P}(uu) \Rightarrow Seq(SS, TT)(rr) = SS(TT(rr)))$

ASSERTIONS

$\forall oo \cdot (oo \in \mathbb{P}(uu) \Rightarrow XX(oo)(\varnothing) = \mathsf{fix}(ff(oo)(\varnothing)))$

END

The Machine *loopterm*

MACHINE
loopterm

SEES
$dovetail, fairloop, propfair, guardfair$

CONSTANTS
pp, oo

PROPERTIES

$pp \subseteq uu \wedge$
$oo \subseteq uu \wedge$

/* WF0 */
$pp \cap cpl(oo) \subseteq FF(pp \cup oo) \wedge$

/* WF1 */
$pp \cap cpl(oo) \subseteq cpl(GG(\varnothing)) \wedge$
$pp \cap cpl(oo) \subseteq GG(oo)$

ASSERTIONS

/* termination of the fair loop */
$oo \cup cpl(GG(\varnothing)) \subseteq XX(oo)(uu)$;

/* total correctness of the fair loop */
$pp \cup oo \subseteq Li(XX(oo))(oo)$;
$pp \cup oo \subseteq XX(oo)(uu)$

END