Chapter 4

A Generalization of Progress

Progress properties constitute a large class of properties frequently encountered in the specification of concurrent systems. Informally characterized as expressing that "*something good will happen*" [Lam77], progress properties are used for specifying or asserting important achievements of concurrent systems such as guarantee of response to service requests, convergence, or absence of starvation. Different from safety properties, which are informally described as expressing that "*nothing bad ever happens*", progress properties inherently refer to infinite program executions in that finite prefixes of program executions do not suffice to characterize them [AS85].

Automatic Checking of Progress Properties is Difficult in Practice

In chapter 3 we have seen how checking safety properties of UNITY programs can be reduced to checking certain local conditions and to finding suitably strong invariants. A similar approach fails in general for progress properties, because such properties cannot be characterized by local checking conditions asserting how pairs of successive states are related to each other (if this were possible then progress properties could be characterized by finite execution prefixes as well); instead, some global variance information needs to be provided in order to measure the progress towards some goal.

There has been significant success with various techniques and methods for mechanical verification or automated verification support, in particular theorem provers and model checkers, in dealing with and in reasoning about safety properties. In practice, these verification methods have been found to be often less useful and attractive when applied to progress properties: establishing progress properties is complicated for theorem proving by the need to perform transfinite induction over a well-founded set, whereas for model checking the alternation depth 2 of characterizations of progress under fairness limits symbolic approaches that are so far the most successful ones. As a consequence, it is not uncommon to formally verify only safety aspects of concurrent systems and to argue about progress properties informally, or even to consider progress properties as practically not very important. With our work we want to help with improving the manageability of proving progress properties while making mechanical checking of progress properties more efficient.

It is our goal in this chapter to propose a new notion of generalized progress properties and to develop a theory for it that makes it possible to derive ordinary[1] progress properties from generalized ones, to utilize design knowledge during reasoning about progress properties, and to improve the performance of automated checking procedures based on the new progress properties.

Generalized Progress Theory Improves Checking Progress Properties

The key idea for accomplishing these goals is to formalize and thereby to make explicit the way in which progress is achieved by a program. We do so by introducing *generalized progress properties* of the form

$$p \overset{W}{\longmapsto} q$$

(pronounced *p leads-to q by W*) for state predicates p and q and for regular ex-

[1]We use the term *ordinary* in connection with progress properties from now on to indicate the traditional notion of progress and to distinguish it from our new notion of *generalized* progress.

pression W over the alphabet of actions of the program under consideration. The regular expression W captures how progress is achieved from states satisfying p to states satisfying q. Using generalized progress properties in the design process for concurrent systems is advantageous in two ways: the designer can incorporate action based design knowledge in the design and verification process, and the actual mechanical verification can possibly be performed more efficiently. The increase in efficiency is due to the fact that a verification system can utilize the information contained in the hints supplied in form of regular expressions in order to eliminate some unnecessary fixpoint computations and to simplify others. The effective use of such action based design knowledge, however, is of great importance beyond improving verifier performance, by making it possible for the designer to provide hints in a formal manner that become part of the verification process and can be used in exploring and debugging programs and specifications. Moreover, the level of provided design knowledge is scalable in the sense that whatever is made available can be used, while without any specific design knowledge the generalized progress properties coincide with the ordinary ones.

The UNITY Advantage

In section 3.1 we have seen how the local checking conditions for safety properties and for basic progress properties were derived from the corresponding proof rules for safety and basic progress of UNITY logic (cf. section 2.2). In a similar fashion the formalization of progress via regular expressions is mostly determined by the UNITY proof rules for leads-to properties: progress by single actions corresponds to **ensures** properties, where the helpful action is named explicitly; sequencing corresponds to transitivity, alternation to finite disjunctions, and repetition to application of leads-to induction. The combination of simplicity of formal specifications and of a rich structure of the deductive system makes UNITY a very suitable basis for the development of the theory of generalized progress and its practical application.

Different Semantics for Generalized Progress

There are different ways for developing and presenting an extension to an existing logic and its semantics: properties can be described operationally in terms of program executions; they can be characterized in relational terms by a set of axioms and proof rules; or they can be characterized by suitable predicate transformers. In our presentation we use different approaches in order to deal with different aspects of the theory in the most suitable way, while demonstrating the close relationship of these approaches to each other.

The *operational characterization* relates generalized progress properties to program executions that often play a significant role in the conception and design of programs, either due to an informal operational specification of the desired program, or as a complementary technique enabling the designer to deal with and to concretize inherently informal aspects of a program specification. An operational semantics is therefore an important tool for capturing and expressing design knowledge.

Defining a property as a *relation over predicates* gives rise to a proof system which is often the most suitable formalism to establish properties of programs under consideration.

Finally, a characterization of generalized progress properties based on *predicate transformers* is appealing because its uniformity makes it suitable for calculational proofs to be used for proving meta-theorems about properties as well as automating proofs of properties for specific programs by model checking.

Organization of This Chapter

The remainder of this chapter is organized as follows: in section 4.1 we introduce the idea of generalized progress properties and relate such properties informally to both program executions and to proofs of ordinary progress properties. In section 4.2 we present a formal semantics for generalized progress properties based on predicate transformers. In section 4.3 we provide a deductive system for proving generalized progress properties of given programs and relate it to predicate transformer semantics. An investigation of the algebraic structure of the family of generalized progress properties for different regular expressions is given in section 4.4. A presentation of an operational semantics for the generalized progress properties in section 4.5 and a discussion of the new theory in section 4.6 conclude the chapter.

4.1 Introduction

Progress properties are generally specified using some temporal operator, be it the *leads-to* operator of UNITY logic, formulae of the form $\mathbf{AG}(p \Rightarrow \mathbf{AF}q)$ of CTL, or formulae like $\mathbf{G}(p \Rightarrow \mathbf{F}q)$ of linear temporal logic. Common to the ways these operators are defined is the fact that they abstract away from how progress is achieved by hiding any reference to individual program actions or variant functions. This abstraction makes it possible to characterize progress simply by pairs of state predicates; however, establishing such progress properties for a given program requires that individual program actions be made explicit (for instance in carrying out a deductive proof), or at least be handled anonymously (for instance in model checking algorithms).

The Advantages of Providing Design Knowledge

Although the formal verification of progress properties is highly desirable, it is often unrealistic to require the designer to provide a *complete* proof of such a property. The level of detail required to carry out a formal proof is often well beyond what is considered practical during the design process; moreover, the designer's expertise might not be sufficient to carry out such a proof efficiently. In spite of these practical difficulties, it is undoubtedly the case that the designer has some (possibly partial) understanding of how progress is achieved by the designed program, either in the form of some operational argument, in the form of a high level proof sketch, or simply based on experience with similar programs. In all these cases a formalism that allows the designer to express such partial, often operational, design knowledge in a simple way and that takes advantage of that knowledge in the verification process, has the potential of improving the effectiveness of the verification task by conducting a proof (by model checking) of some properties without forcing the designer to supply a complete proof herself.

If the partial knowledge supplied by the designer is sufficient to derive such a proof efficiently, the interactive verification has been completed successfully. If such a proof cannot be found efficiently (or cannot be found at all), the designer needs to be assisted in supplying further (more specific) design knowledge. If she cannot provide such knowledge, or if the provided knowledge causes any inconsistency, appropriate debugging information should be made available to aid in either debugging the design, or in pointing out parts that require additional design knowledge in order to be verified efficiently. However, not more information should be asked for from the designer than what is necessary for establishing the required properties.

By providing a way of making this form of design knowledge formally accessible to reasoning about programs, and by taking advantage of such knowledge in the verification process, we could meet our goals of providing a more powerful and effective way of dealing with progress properties than by the use of the leads-to operator alone.

An Example

In the following we present a small example to illustrate the idea of how progress properties are generalized by including explicit action-based progress information. We then argue how this new theory and its associated methodology can be used in program verification.

Let us consider the following program *UpDown*, in which n is an integer counter, that can always be decremented, but can only be incremented if the boolean variable b is false:

program *UpDown*

declare

var n : *integer*

var b : *boolean*

assign

[up] $\quad n := n + 1 \quad$ **if** $\neg b$

[down] $\quad n := n - 1$

[set] $\quad b :=$ true

end

A progress property of program *UpDown* is that in any execution eventually n becomes negative, which is expressed by the following UNITY leads-to property:

$$\text{true} \quad \mapsto \quad n < 0.$$

The following informal argument serves as a justification for this claim: given any execution and any state reached during the execution in which n is not negative, there is some subsequent state in which b holds: at the latest the first state after an execution of [set] [2] satisfies b, due to the unconditional fairness there always is such a next occurrence of [set], and no other action falsifies b. In such a state satisfying b, either n is negative or a finite number of executions of [down] (namely, one more than the value of n in that state) makes n negative. Again, due to the unconditional fairness there are sufficiently many such occurrences of [down], and no other action interferes with [down] by increasing n or falsifying b. Therefore, for any execution and any state there is a future state[3] satisfying $n < 0$.

The above operational argument suggests that one can think of the progress from true to $n < 0$ as being achieved by virtue of a strategy expressed by the regular expression

[2] Square brackets are used in the following as part of the action labels.

[3] possibly the current state

$$[\text{set}][\text{down}]^*,$$

over the alphabet of actions of *UpDown*, i.e., by one [set] action followed by some finite number of [down] actions. Using the notation for generalized progress properties, we can state that program *UpDown* satisfies the property

$$\text{true} \overset{[\text{set}][\text{down}]^*}{\longmapsto} n < 0.$$

Before we formalize the notion of a strategy for achieving progress and define the operational semantics of generalized progress properties, we show that there is not only an operational interpretation of such regular expression strategies, but also a close connection to proofs in the deductive system of UNITY logic.

To this end we present a proof in UNITY logic of the property true $\mapsto$ $n < 0$ of program *UpDown*:

0. true **ensures** b

   ; from program text via [set]

1. true $\mapsto$ b

   ; promotion from 0

2. $b \wedge n = k$ **ensures** $b \wedge n < k$

   ; from program text via [down]

3. $b \wedge n = k \mapsto b \wedge n < k$

   ; promotion from 2

4. $b \wedge |n| = k \mapsto (b \wedge |n| < k) \vee n < 0$

   ; arithmetic case split and disjunction on 3

5. $b \mapsto n < 0$

   ; leads-to induction on 4 with metric $|n|$ over the naturals

6. true $\mapsto$ $n < 0$

   ; transitivity with 1 and 5

Steps 0 and 1 of this proof correspond to the [set] action in our proposed strategy; similarly steps 2 and 3 correspond to the [down] action. The inductive argument of steps 4 and 5 establishes the progress via $[\text{down}]^*$; finally, step 6, combines the substrategies by sequencing. We therefore can argue that the above proof has a structure corresponding to the regular expression $[\text{set}][\text{down}]^*$. However, this regular expression is much less detailed than the complete proof: in particular, the state predicates needed to combine the different parts of the proof together, are omitted. Hence, even a general idea about the structure of a proof of a progress property can be turned into a strategy without requiring a complete and detailed proof.

Some Ways for Providing Design Knowledge

In summary, we have suggested that there are two ways in which the designer of a concurrent system could formulate her design knowledge and come up with strategies in the form of regular expressions for progress properties of the program under consideration: suitable regular expressions can either be proposed based on the operational understanding of the program under construction, or can be derived from a high-level approximation of a deductive proof of the property. Both techniques help the designer to express design knowledge and to convey it to a verifier that can take advantage of the information provided. Moreover, the expressed design knowledge can play an important role in reasoning about the program, in testing hypotheses, and in debugging both programs and properties.

In the rest of this chapter we develop the theory of generalized progress. The practical application of this theory to model checking, as well as a demonstration of the techniques for utilizing design knowledge, follows in chapter 5.

4.2 A Predicate Transformer Semantics

Predicate transformers have been successfully used for defining and reasoning about semantics of both sequential ([Dij76, DS90]) and concurrent ([JKR89, Kna92]) programs. In this section we present a predicate transformer semantics for the generalized progress properties and use it for investigating some fundamental characteristics of these properties.

We are interested in a predicate transformer semantics for several reasons: the use of predicate transformers allows us to stay completely in the realm of predicate calculus, instead of having to introduce axioms for each relation of a deductive system, thereby facilitating a calculational style of reasoning; certain characteristics of the generalized leads-to operator, such as monotonicity and continuity, can be stated and answered more easily using predicate transformers; moreover, the predicate transformers prove to be invaluable for investigating the algebraic structure of the generalized progress properties (cf. section 4.4); and finally, the fixpoint definitions of the predicate transformers for generalized progress give rise to symbolic algorithms which we subsequently use in our new model checking procedures for progress properties in chapter 5.

The treatment of the predicate transformers for generalized progress is organized as follows: in section 4.2.1 we introduce a family of predicate transformers **wltr** (for *weakest leads-to by regular expression*) by giving a fixpoint characterization and exhibiting some basic properties of them. In section 4.2.2, we investigate the junctivity properties of the **wltr** predicate transformers. In section 4.2.3 we establish the close connection between the **wltr** predicate transformers and the **wlt** predicate transformer (for *weakest leads-to*, [JKR89]) characterizing the ordinary leads-to properties of UNITY logic.

4.2.1 Predicate Transformers for Generalized Progress

We begin the formal treatment of the semantics of generalized progress properties by defining a family of predicate transformers $\mathbf{wltr}.W$ for any W in $\mathcal{R}_F$ inductively over the structure of W:

Definition 2 *For any W in $\mathcal{R}_F$, the predicate transformer* $\mathbf{wltr}.W$ *is defined inductively over the structure of W such that for all α in $F.A$, all U, V in $\mathcal{R}_F$, and all state predicates q in $\mathcal{P}_F$:*

$$[\![\mathbf{wltr}.\varepsilon.q \quad \equiv \quad q]\!] \qquad \textbf{\textit{(wltrEps)}}$$

$$[\![\mathbf{wltr}.\alpha.q \quad \equiv \quad \langle \nu Z :: (\mathbf{wco}.(Z \vee q) \wedge \mathbf{wp}.\alpha.q) \vee q\rangle]\!] \qquad \textbf{\textit{(wltrAct)}}$$

$$[\![\mathbf{wltr}.(UV).q \quad \equiv \quad \mathbf{wltr}.U.(\mathbf{wltr}.V.q)]\!] \qquad \textbf{\textit{(wltrSeq)}}$$

$$[\![\mathbf{wltr}.(U+V).q \quad \equiv \quad \mathbf{wltr}.U.q \ \vee \ \mathbf{wltr}.V.q]\!] \qquad \textbf{\textit{(wltrAlt)}}$$

$$[\![\mathbf{wltr}.U^*.q \quad \equiv \quad \langle \mu Z :: q \ \vee \ \mathbf{wltr}.U.Z\rangle]\!] \qquad \textbf{\textit{(wltrStar)}}$$

Informally, the predicate $\mathbf{wltr}.W.q$ is intended to characterize all those states from which any execution characterized by W leads to a state satisfying q, where the notion of executions characterized by regular expressions will be made precise later in section 4.5.

Even with a very informal understanding of progress characterized by regular expressions, we can attempt to motivate the above definitions. To this end we note that **(wltrEps)** captures precisely the notion of progress without actions, where the start predicate is the goal predicate, that **(wltrSeq)** captures the notion of sequencing which corresponds to the functional composition of the predicate transformers of the subexpressions, and that **(wltrAlt)** captures the notion of choice which corresponds to the disjunction of the predicate transformers of the subexpressions.

The form of **(wltrAct)** is suggested by observing that for any action α in $\mathcal{R}_F$ we expect $\mathbf{wltr}.\alpha.q$ to be the weakest predicate that ensures q via α, i.e.,

$$\mathbf{wltr}.\alpha.q \ \mathbf{ensures}_\alpha \ q \qquad \textbf{(E0)}$$

$$Z \ \mathbf{ensures}_\alpha \ q \quad \Rightarrow \quad [Z \Rightarrow \mathbf{wltr}.\alpha.q] \qquad \textbf{(E1)}$$

Manipulating the definition of $\mathbf{ensures}_\alpha$ we observe

$$Z \ \mathbf{ensures}_\alpha \ q$$

$$\equiv \quad \{\text{definition of } \mathbf{ensures}_\alpha\}$$

$$[Z \wedge \neg q \Rightarrow \mathbf{wco}.(Z \vee q)] \wedge [Z \wedge \neg q \Rightarrow \mathbf{wp}.\alpha.q]$$
$$\equiv \quad \{\text{shunting, predicate calculus}\}$$
$$[Z \Rightarrow ((\mathbf{wco}.(Z \vee q) \wedge \mathbf{wp}.\alpha.q) \vee q)]$$

In order to characterize the weakest predicate that ensures q via α it therefore seems reasonable to ask for the weakest solution of the following equation:

$$Z : [\![Z \;\Rightarrow\; ((\mathbf{wco}.(Z \vee q) \wedge \mathbf{wp}.\alpha.q) \vee q)]\!]$$

The righthand side of this equation is monotonic in Z and therefore (by the Knaster-Tarski Theorem) has a weakest solution, which is the same as the weakest solution of the equation

$$Z : [\![Z \;\equiv\; ((\mathbf{wco}.(Z \vee q) \wedge \mathbf{wp}.\alpha.q) \vee q)]\!]$$

which we denote by $\langle \nu Z :: (\mathbf{wco}.(Z \vee q) \wedge \mathbf{wp}.\alpha.q) \vee q \rangle$ and which we use as our definition of $\mathbf{wltr}.\alpha.q$. We note that by virtue of the above construction and by the relationship between the everywhere operators it follows that $\mathbf{wltr}.\alpha.q$ satisfies **(E0)** and **(E1)**.

For **(wltrStar)** we expect that $\mathbf{wltr}.U^*.q$ be the weakest predicate that leads to q by some finite sequence of executions of U. Together with the observation that $\mathbf{wltr}.U^*.q$ certainly is at least as weak as q, this makes it seem reasonable to ask for a solution of the equation

$$Z : [\![Z \;\equiv\; q \vee \mathbf{wltr}.U.Z]\!]$$

Due to the required finiteness of the repetitions we need to consider the strongest solution of the above equation. In the next paragraph, we establish that for any W in $\mathcal{R}_F$ $\mathbf{wltr}.W$ is a monotonic predicate transformer, by induction on the structure of W. It follows that the righthand side of the above equation is monotonic in Z, and that therefore (again by the theorem of Knaster-Tarski) the equation has indeed a least fixpoint which we denote by $\langle \mu Z :: q \vee \mathbf{wltr}.U.Z \rangle$ and which we use as our definition of $\mathbf{wltr}.U^*.q$.

As some important properties of the **wltr** predicate transformers we establish that each such predicate transformer is monotonic, strict, and weakening:

Lemma 6 (Basic Properties of wltr) *For any W in $\mathcal{R}_F$ and any p, q in $\mathcal{P}_F$:*

$$[\![p \Rightarrow q]\!] \;\Rightarrow\; [\![\mathbf{wltr}.W.p \Rightarrow \mathbf{wltr}.W.q]\!] \qquad \textit{\textbf{(wltrMon)}}$$

$$[\![\mathbf{wltr}.W.\text{false} \;\equiv\; \text{false}]\!] \qquad \textit{\textbf{(wltrStrict)}}$$

$$[\![q \Rightarrow \mathbf{wltr}.W.q]\!] \qquad \textit{\textbf{(wltrWeaken)}}$$

Proof . We prove each property by induction over the structure of W. For **(wltrMon)** we observe the following: $\mathbf{wltr}.\varepsilon$ is the identity function and is therefore obviously monotonic; $\mathbf{wltr}.\alpha$ is defined as a fixpoint of a monotonic equation and is therefore monotonic itself; both $\mathbf{wltr}.(UV)$ and $\mathbf{wltr}.(U+V)$ are defined by monotonic functions of the (by the induction hypothesis) monotonic predicate transformers $\mathbf{wltr}.U$ and $\mathbf{wltr}.V$; and $\mathbf{wltr}.U^*$ is defined as a fixpoint of a monotonic equation using the (by the induction hypothesis) monotonic predicate transformer $\mathbf{wltr}.U$, and is therefore monotonic as well.

For **(wltrStrict)** we show only the cases corresponding to single actions and to repetition, the remaining cases are trivial. We observe for any α in $F.A$:

$$\begin{array}{ll}
 & \mathbf{wltr}.\alpha.\text{false} \\
\equiv & \{\mathbf{(wltrAct)}\} \\
 & \langle \nu Z :: (\mathbf{wco}.(p \vee \text{false}) \wedge \mathbf{wp}.\alpha.\text{false}) \vee \text{false}\rangle \\
\equiv & \{\mathbf{wp}.\alpha \text{ is strict, predicate calculus}\} \\
 & \langle \nu Z :: \text{false}\rangle \\
\equiv & \{\text{predicate calculus}\} \\
 & \text{false}
\end{array}$$

and for any W in $\mathcal{R}_F$:

$$\mathbf{wltr}.W^*.\text{false}$$

$\equiv$ {**(wltrStar)**}

$\langle \mu Z :: \text{false} \ \vee \ \mathbf{wltr}.U.Z \rangle$

$\equiv$ {induction hypothesis, predicate calculus}

false

Similarly, the proof for **(wltrWeaken)** is straight forward if we use the fact that in the cases corresponding to single actions and repetition any fixpoint of the respective equations is implied by q.

End of Proof.

4.2.2 Junctivity Properties of the wltr Predicate Transformers

We have already established in the previous section that for each W in $\mathcal{R}_F$ the predicate transformer $\mathbf{wltr}.W$ is monotonic. In this section we show that in general $\mathbf{wltr}.W$ does not enjoy other interesting junctivity properties. In particular we demonstrate that $\mathbf{wltr}.W$, where W contains some action from $F.A$, is neither finitely disjunctive nor finitely conjunctive, that it is not or-continuous, and is and-continuous only if the regular expressions W does not contain the * operator (see section 2.1.4 for the definition of junctivity properties).

We start with the results about and-continuity:

Lemma 7 *For any W in $\mathcal{R}_F$ not containing the repetition operator *,* $\mathbf{wltr}.W$ *is and-continuous. Allowing the * operator in general destroys and-continuity.*

Proof . The proof proceeds by induction on the structure of W. We show that all operators but the * operator preserve and-continuity, and that the * operator in general destroys it.

Since $[\![\mathbf{wltr}.\varepsilon.q \equiv q]\!]$ by **(wltrEps)**, $\mathbf{wltr}.\varepsilon$ is trivially and-continuous. By **(wltrAct)**, $\mathbf{wltr}.\alpha.q$ for some α in $F.A$ is the weakest solution of the equation

$$X : [\![X \ \equiv \ \tau.X.q]\!]$$

where

$$[\![\tau.X.Y \;\equiv\; (\mathbf{wco}.(X \vee Y) \wedge \mathbf{wp}.\alpha.Y) \vee Y]\!]$$

From the universal conjunctivity of **wco** and **wp** and the and-continuity of disjunction it follows that τ is and-continuous. Hence, by theorem 2, $\mathbf{wltr}.\alpha$ is and-continuous as well. Function composition and disjunction preserve and-continuity, hence for and-continuous $\mathbf{wltr}.U$ and $\mathbf{wltr}.V$ both $\mathbf{wltr}.UV$ and $\mathbf{wltr}.(U+V)$ are and-continuous as well.

To show that the * operator does not preserve and-continuity we consider the program

program *AndContinuity*
  declare
    var n : *integer*
  assign
    $[\alpha]$ $\qquad n := n+1$
end

and define for any natural i the predicate $Q.i$ by $[\![Q.i \equiv n \geq i]\!]$. Clearly, $\langle \forall i, j :: i \leq j \Rightarrow [\![Q.i \Leftarrow Q.j]\!]\rangle$, hence $\{i : i \in \mathbf{N} : Q.i\}$ is linear. For this set of predicates we observe that

$$\begin{array}{ll}
 & \mathbf{wltr}.\alpha^*.\langle \forall i : i \in \mathbf{N} : Q.i\rangle \\
\equiv & \{\text{definition of } Q.i\text{, arithmetic}\} \\
 & \mathbf{wltr}.\alpha^*.\text{false} \\
\equiv & \{(\mathbf{wltrStrict})\} \\
 & \text{false}
\end{array}$$

but that

$$\begin{array}{ll}
 & \langle \forall i : i \in \mathbf{N} : \mathbf{wltr}.\alpha^*.(Q.i)\rangle \\
\equiv & \{\text{calculation using } (\mathbf{wltrStar}),\ (\mathbf{wltrAct})\}
\end{array}$$

$$\langle \forall i : i \in \mathbf{N} : \text{true} \rangle$$

$\equiv$ {predicate calculus}

true

End of Proof.

The remaining results are presented in the following lemma:

Lemma 8 *For any W in $\mathcal{R}_F$ not containing actions from $F.A$,* $\mathbf{wltr}.W$ *is both universally disjunctive and universally conjunctive. For any W containing at least one action from $F.A$,* $\mathbf{wltr}.W$ *is in general*

(0) *not or-continuous,*

(1) *not finitely disjunctive,*

(2) *not finitely conjunctive.*

Proof . It is easily seen by induction on the structure of W, that $[\![\mathbf{wltr}.W.q \equiv q]\!]$ for any W built from ε, sequencing, alternation, and repetition alone. For such W, $\mathbf{wltr}.W$ is therefore clearly universally disjunctive and universally conjunctive.
For the three negative results we exhibit counterexamples:

ad **(0)**: Consider the program

program *OrContinuity*
  declare
    var n : *integer*
    var $d : \{-1, 1\}$
  assign
    $[\alpha]$ $\quad n, d := n + d, -1$
    $[\beta]$ $\quad n := n + d$
end

which is a slight modification of the program of section 4.1.

For each natural i define $Q.i$ by $[\![Q.i \;\equiv\; n \leq 0 \vee (d = -1 \wedge n \leq i)]\!]$. Clearly, $\langle \forall i, j :: i \leq j \Rightarrow [\![Q.i \Rightarrow Q.j]\!] \rangle$, hence $\{i : i \in \mathbf{N} : Q.i\}$ is linear. Furthermore, it can be shown that $[\![\mathbf{wltr}\,.\alpha.(Q.i) \equiv Q.(i+1)]\!]$. From this it follows that

$$\begin{array}{ll} & \mathbf{wltr}\,.\alpha.\langle \exists i : i \in \mathbf{N} : Q.i \rangle \\ \equiv & \{\text{definition of } Q.i\text{, arithmetic}\} \\ & \mathbf{wltr}\,.\alpha.(n \leq 0 \vee d = -1) \\ \equiv & \{\textbf{(wltrAct)}\text{, from program}\} \\ & \text{true} \end{array}$$

but that

$$\begin{array}{ll} & \langle \exists i : i \in \mathbf{N} : \mathbf{wltr}\,.\alpha.(Q.i) \rangle \\ \equiv & \{\textbf{(wltrAct)}\text{, see above}\} \\ & \langle \exists i : i \in \mathbf{N} : Q.(i+1) \rangle \\ \equiv & \{\text{definition of } Q.i\text{, arithmetic}\} \\ & n \leq 0 \vee d = -1 \end{array}$$

ad **(1)**: Consider the program

program *FiniteDisjunctivity*

  declare

    var $n : \{0, 1, 2, 3, 4\}$

  initially

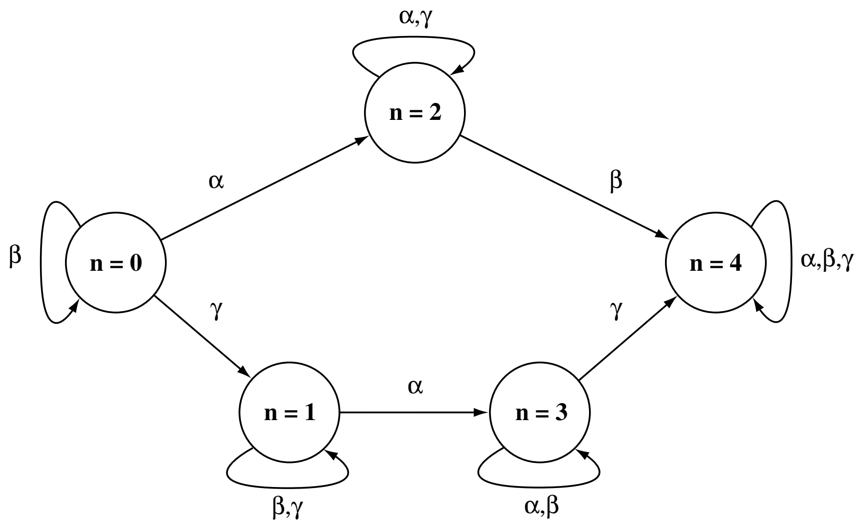
    $n = 0$

  assign

$[\alpha]$	$n := n + 2$	**if** $n < 2$
$[\beta]$	$n := 4$	**if** $n = 2$
$[\gamma]$	$n := n + 1$	**if** $n = 0 \vee n = 3$

end

which has the following state-transition diagram:



For the predicates p and q defined by $[\![p \equiv n = 2 \vee n = 4]\!]$ and $[\![q \equiv n = 3 \vee n = 4]\!]$ we observe that

$$
\begin{array}{ll}
& \mathbf{wltr}.\alpha.(p \vee q) \\
\equiv & \{\text{definition of } p,\, q\} \\
& \mathbf{wltr}.\alpha.(n = 2 \vee n = 3 \vee n = 4) \\
\equiv & \{\text{calculation using } \mathbf{(wltrAct)}\} \\
& \text{true}
\end{array}
$$

but that

$$
\begin{array}{ll}
& \mathbf{wltr}.\alpha.p \ \vee \ \mathbf{wltr}.\alpha.q \\
\equiv & \{\text{calculations using } \mathbf{(wltrAct)}\} \\
& (n = 2 \vee n = 4) \ \vee \ (n = 1 \vee n = 3 \vee n = 4) \\
\equiv & \{\text{predicate calculus}\} \\
& n \neq 0
\end{array}
$$

ad **(2)**: Consider the program

program *FiniteConjunctivity*

  declare

var b : *boolean*

assign

$[\alpha]$ $\qquad b := \neg b$

end

and observe for the predicates b and $\neg b$ that

$$\begin{array}{ll} & \mathbf{wltr}.\alpha.(b \wedge \neg b) \\ \equiv & \{\text{predicate calculus}\} \\ & \mathbf{wltr}.\alpha.\,\text{false} \\ \equiv & \{(\mathbf{wltrStrict})\} \\ & \text{false} \end{array}$$

but that

$$\begin{array}{ll} & \mathbf{wltr}.\alpha.b \;\wedge\; \mathbf{wltr}.\alpha.(\neg b) \\ \equiv & \{\text{calculations using } (\mathbf{wltrAct})\} \\ & \text{true} \wedge \text{true} \\ \equiv & \{\text{predicate calculus}\} \\ & \text{true} \end{array}$$

End of Proof.

4.2.3 Relating wltr to wlt

We are now in the position to answer the important question about the relationship between the ordinary leads-to properties of UNITY logic characterized by the **wlt** predicate transformer and our generalized progress properties. In the following we establish the relationship between **wltr** and **wlt**, later in section 4.3.3 we use the result obtained here to state the close connection of the proof systems for ordinary and generalized progress properties.

The relationship between **wltr** and **wlt** is characterized by the following theorem:

Theorem 9 (Relating wltr and wlt) *For any W in $\mathcal{R}_F$, and state predicate q in $\mathcal{P}_F$:*

$$[\![\mathbf{wltr}.W.q \;\Rightarrow\; \mathbf{wlt}.q]\!] \qquad \textbf{\textit{(wltrSound)}}$$

$$\langle \exists W : W \in \mathcal{R}_F : [\![\mathbf{wlt}.q \;\equiv \mathbf{wltr}.W.q]\!]\rangle \qquad \textbf{\textit{(wltrCompl)}}$$

The first part of the theorem can be referred to as a *soundness* result since it asserts that states from which a q state is reached by virtue of W are states from which a q state is reached eventually in the sense of ordinary progress. Conversely, the second part can be seen as a *completeness* result since it shows that any state from which a q state is reached eventually is a state from which a q state is reached by virtue of some regular expression W in $\mathcal{R}_F$.

In order to prove these results we recall a few properties of the **wlt** predicate transformer, either taken from [JKR89] or being simple consequences of results found there. First, **wlt** has the following fixpoint characterization:

$$[\![\mathbf{wlt}.q \;\equiv\; \langle \mu Z :: q \vee \mathbf{we}.Z\rangle]\!] \qquad \textbf{(wlt0)}$$

$$[\![\mathbf{we}.q \;\equiv\; \langle \exists \alpha : \alpha \in F.A : \mathbf{stp}.\alpha.q\rangle]\!] \qquad \textbf{(wlt1)}$$

$$[\![\mathbf{stp}.\alpha.q \;\equiv\; \langle \nu Z :: (\mathbf{wco}.Z \wedge \mathbf{wp}.\alpha.q) \vee q\rangle]\!] \qquad \textbf{(wlt2)}$$

Furthermore, **wlt** and **stp** enjoy the following properties:

$$[\![q \;\Rightarrow\; \mathbf{wlt}.q]\!] \qquad \textbf{(wlt3)}$$

$$[\![\mathbf{stp}.\alpha.q \;\Rightarrow\; \mathbf{wlt}.q]\!] \qquad \textbf{(wlt4)}$$

$$[\![\mathbf{wlt}.(\mathbf{wlt}.q) \;\equiv\; \mathbf{wlt}.q]\!] \qquad \textbf{(wlt5)}$$

We also need the following lemma relating **stp** and **wltr**:

Lemma 10 *For a program F and any action α in $F.A$:*

$$[\![\mathbf{stp}.\alpha \;\equiv\; \mathbf{wltr}.\alpha]\!]$$

Proof . $\mathbf{stp}.\alpha.q$ is by **(wlt2)** the weakest solution of the equation $X : [\![X \equiv f.X]\!]$ with f defined by

$$[\![f.X \;\equiv\; (\mathbf{wco}.X \wedge \mathbf{wp}.\alpha.q) \vee q]\!]$$

and $\mathbf{wltr}.\alpha.q$ is by **(wltrAct)** the weakest solution of the equation $X : [\![X \equiv g.X]\!]$ with g defined by

$$[\![g.X \;\equiv\; (\mathbf{wco}.(X \vee q) \wedge \mathbf{wp}.\alpha.q) \vee q]\!]$$

Clearly both f and g are monotonic. Since **wco** is monotonic we have $[\![f \Rightarrow g]\!]$, and hence by theorem 3 that $[\![\mathbf{stp}.\alpha \Rightarrow \mathbf{wltr}.\alpha]\!]$.
For the converse, we observe that $\mathbf{wltr}.\alpha$, as a fixpoint of g, satisfies

$$[\![\mathbf{wltr}.\alpha.q \;\equiv\; (\mathbf{wco}.(\mathbf{wltr}.\alpha.q \vee q) \wedge \mathbf{wp}.\alpha.q) \vee q]\!] \qquad \mathbf{(0)}$$

and that $\mathbf{stp}.\alpha$, as the greatest fixpoint of f, satisfies

$$[\![X \equiv f.X]\!] \Rightarrow [\![X \Rightarrow \mathbf{stp}.\alpha.q]\!] \qquad \mathbf{(1)}$$

with which we have for all q:

$$
\begin{array}{ll}
 & [\![\mathbf{wltr}.\alpha.q \;\Rightarrow\; \mathbf{stp}.\alpha.q]\!] \\
\Leftarrow & \{\mathbf{(1)}\} \\
 & [\![\mathbf{wltr}.\alpha.q \;\equiv\; f.(\mathbf{wltr}.\alpha.q)]\!] \\
\equiv & \{[\![\mathbf{wltr}.\alpha.q \equiv q \vee \mathbf{wltr}.\alpha.q]\!] \text{ from } \mathbf{(0)}, \text{ definition of } f\} \\
 & [\![\mathbf{wltr}.\alpha.q \;\equiv\; (\mathbf{wco}.(\mathbf{wltr}.\alpha.q \vee q) \wedge \mathbf{wp}\,\alpha.q) \vee q]\!] \\
\equiv & \{\mathbf{(0)}\} \\
 & \text{true}
\end{array}
$$

End of Proof.

We are now ready to prove the soundness and completeness results.
Proof of theorem 9. For **(wltrSound)** we observe for all q, and W by induction over the structure of W:

case $W = \varepsilon$:

$$\begin{array}{ll} & \mathbf{wltr}.\varepsilon.q \\ \equiv & \{(\mathbf{wltrEps})\} \\ & q \\ \Rightarrow & \{(\mathbf{wlt3})\} \\ & \mathbf{wlt}.q \end{array}$$

case $W = \alpha$ for some $\alpha \in F.A$:

$$\begin{array}{ll} & \mathbf{wltr}.\alpha.q \\ \equiv & \{\text{lemma above}\} \\ & \mathbf{stp}.\alpha.q \\ \Rightarrow & \{(\mathbf{wlt4})\} \\ & \mathbf{wlt}.q \end{array}$$

case $W = UV$:

$$\begin{array}{ll} & \mathbf{wltr}.UV.q \\ \equiv & \{(\mathbf{wltrSeq})\} \\ & \mathbf{wltr}.U.(\mathbf{wltr}.V.q) \\ \Rightarrow & \{\text{induction hypothesis, twice, } \mathbf{wltr}.U \text{ is monotonic}\} \\ & \mathbf{wlt}.(\mathbf{wlt}.q) \\ \equiv & \{(\mathbf{wlt5})\} \\ & \mathbf{wlt}.q \end{array}$$

case $W = U + V$:

$$\begin{array}{ll} & \mathbf{wltr}.(U+V).q \\ \equiv & \{(\mathbf{wltrAlt})\} \\ & \mathbf{wltr}.U.q \vee \mathbf{wltr}.V.q \\ \Rightarrow & \{\text{induction hypothesis, twice, predicate calculus}\} \\ & \mathbf{wlt}.q \end{array}$$

case $W = U^*$:

$$
\begin{array}{ll}
 & [\![\mathbf{wltr}\,.U^*.q \;\Rightarrow\; \mathbf{wlt}\,.q]\!] \\
\equiv & \{\mathbf{(wltrStar)}\} \\
 & [\![\langle \mu Z :: q \vee \mathbf{wltr}\,.U.Z\rangle \;\Rightarrow\; \mathbf{wlt}\,.q]\!] \\
\Leftarrow & \{\text{induction hypothesis, theorem 3, predicate calculus}\} \\
 & [\![\langle \mu Z :: q \vee \mathbf{wlt}\,.Z\rangle \;\Rightarrow\; \mathbf{wlt}\,.q]\!] \\
\Leftarrow & \{\text{least fixpoint property}\} \\
 & [\![q \vee \mathbf{wlt}\,.(\mathbf{wlt}\,.q) \;\equiv\; \mathbf{wlt}\,.q]\!] \\
\equiv & \{\mathbf{(wlt3)},\ \mathbf{(wlt5)},\ \text{predicate calculus}\} \\
 & \text{true}
\end{array}
$$

For **(wltrCompl)** it suffices to exhibit a regular expression C in $\mathcal{R}_F$ and to demonstrate that for C the implication $[\![\mathbf{wlt} \Rightarrow \mathbf{wltr}\,.C]\!]$ holds. Let L be a sequence of all elements of $F.A$ and define C as

$$C \;=\; (\langle + i : L : L.i\rangle)^*.$$

Using **(wltrEps)**, **(wltrAlt)**, and **(wltrStar)** it is easily seen that

$$[\![\mathbf{wltr}\,.(\langle + i : L : L.i\rangle).q \;\equiv\; \langle \exists \alpha : \alpha \in F.A : \mathbf{wltr}\,.\alpha.Z\rangle]\!]$$
$$[\![\mathbf{wltr}\,.C.q \;\equiv\; \langle \mu Z :: q \vee \langle \exists \alpha : \alpha \in F.A : \mathbf{wltr}\,.\alpha.Z\rangle\rangle]\!]$$

From $[\![\mathbf{wltr}\,.\alpha \equiv \mathbf{stp}\,.\alpha]\!]$ (by the lemma above) and **(wlt0)**, and **(wlt1)**, we see that indeed $[\![\mathbf{wltr}\,.C \equiv \mathbf{wlt}]\!]$.

End of Proof.

4.3 Reasoning about Generalized Progress

In the previous section we have introduced the notion of generalized progress properties of a program F by defining a family of predicate transformers **wltr** characterizing progress by regular expressions. While such a semantics is very useful for investigating such properties and reasoning about them in a general way, it is often more practical to carry out proofs of progress properties of specific programs using a deductive proof system. It is therefore our goal for this section to present such a proof system for generalized progress properties, to exhibit some useful proof rules, and to establish its connection to the previously presented predicate transformer semantics.

We achieve this goal in three steps: in section 4.3.1 we introduce the *generalized leads-to* relation over regular expressions and pairs of state predicates as the strongest relation generated by a set of inference rules; in section 4.3.2 we show how closely the generalized leads-to relation and the **wltr** predicate transformers are related. Finally, we list some useful proof rules in section 4.3.3.

4.3.1 A Deductive System

For a given program F we define in the following a ternary relation $(\lambda W.\lambda p.\lambda q.\, p \overset{W}{\longmapsto} q)$ (pronounced *p leads-to q by W*) over state predicates p and q in $\mathcal{P}_F$ and regular expressions W in $\mathcal{R}_F$ as the strongest relation generated by a set of inference rules. Such a characterization in terms of inference rules is well suited to carrying out proofs of properties of specific programs.

In order to state the proof rules we need to introduce the notion of a *metric* over the reachable state space of F. In the following we use **Ord** to denote the set of ordinal numbers.

Definition 3 (Metric) *A* metric M *for a given program* F *is a family of state predicates* $\{i : i \in \mathbf{Ord} : M.i\}$ *from* $\mathcal{P}_F$*, such that the following two conditions are met:*

$$[\langle \exists i : i \in \mathbf{Ord} : M.i \rangle] \qquad \textit{\textbf{(MetricExh)}}$$

$$\langle \forall i, j : i \in \mathbf{Ord} \wedge j \in \mathbf{Ord} : i \neq j \Rightarrow [\neg(M.i \wedge M.j)] \rangle \qquad \textit{\textbf{(MetricDis).}}$$

The first condition states that the predicates in M exhaust the reachable state space of F, the second asserts that any two predicates with different indices are disjoint.

The predicates in M are totally ordered by the ordering relation $\preceq$ (pronounced *precedes*) obtained from lifting the total order relation on the ordinal indices to the predicates:

$$\langle \forall i, j : i \in \mathbf{Ord} \land j \in \mathbf{Ord} : i \leq j \;\equiv\; M.i \preceq M.j \rangle$$

After these remarks we are ready to define the generalized leads-to relation in terms of a set of generating inference rules as follows:

Definition 4 (Generalized Leads-To Relation) *For a given program F the relation $(\lambda W.\lambda p.\lambda q.\; p \stackrel{W}{\longmapsto} q)$ is the smallest subset of $\mathcal{R}_F \times \mathcal{P}_F \times \mathcal{P}_F$ satisfying the following inference rules for all regular expressions U, V in $\mathcal{R}_F$, for all actions α in $F.A$, for all state predicates p, p', q, r, and s in $\mathcal{P}_F$, and for all metrics M for F:*

$$\frac{[p \Rightarrow q]}{p \stackrel{\varepsilon}{\longmapsto} q} \qquad \textbf{\textit{(PrEps)}}$$

$$\frac{[p \Rightarrow p'], \quad p' \;\mathbf{ensures}_\alpha\; q}{p \stackrel{\alpha}{\longmapsto} q} \qquad \textbf{\textit{(PrAct)}}$$

$$\frac{p \stackrel{U}{\longmapsto} r, \quad r \stackrel{V}{\longmapsto} q}{p \stackrel{UV}{\longmapsto} q} \qquad \textbf{\textit{(PrSeq)}}$$

$$\frac{p \stackrel{U}{\longmapsto} q, \quad r \stackrel{V}{\longmapsto} q}{p \lor r \stackrel{U+V}{\longmapsto} q} \qquad \textbf{\textit{(PrAlt)}}$$

$$\frac{[p \Rightarrow p'], \langle \forall i : i \in \mathbf{Ord} : p' \land M.i \stackrel{U}{\longmapsto} (p' \land \langle \exists j : j < i : M.j \rangle) \lor q \rangle}{p \stackrel{U^*}{\longmapsto} q} \qquad \textbf{\textit{(PrStar)}}$$

An important observation is that for any W in $\mathcal{R}_F$ there is exactly one inference rule that can be used for establishing that a triple (W, p, q) satisfies the relation. It follows that the above rules are actually equivalences rather than implications[4]. For instance, if $p \stackrel{U^*}{\longmapsto} q$ holds for a program F, then there exists a metric M and

[4]The lack of such equivalences for the ordinary leads-to relation was identified as the main shortcoming of leads-to in section 3.

a state predicate p' at least as weak as p, such that for all ordinals i, $p' \wedge M.i \overset{U}{\longmapsto} (p' \wedge \langle \exists j : j < i : M.j \rangle) \vee q$ holds,

We can therefore render the above proof rules in the form of the following equivalences:

$$
\begin{array}{lcll}
p \overset{\varepsilon}{\longmapsto} q & \equiv & [p \Rightarrow q] & \textbf{(AxEps)} \\
p \overset{\alpha}{\longmapsto} q & \equiv & \langle \exists p' : [p \Rightarrow p'] : p' \ \textbf{ensures}_\alpha \ q \rangle & \textbf{(AxAct)} \\
p \overset{UV}{\longmapsto} q & \equiv & \langle \exists r :: (p \overset{U}{\longmapsto} r) \wedge (r \overset{V}{\longmapsto} q) \rangle & \textbf{(AxSeq)} \\
p \overset{U+V}{\longmapsto} q & \equiv & \langle \exists r, s : [r \vee s \equiv p] : (r \overset{U}{\longmapsto} q) \wedge (s \overset{V}{\longmapsto} q) \rangle & \textbf{(AxAlt)} \\
p \overset{U^*}{\longmapsto} q & \equiv & \langle \exists p' : [p \Rightarrow p'] : \langle \exists M : M \text{ is metric} : & \textbf{(AxStar)} \\
 & & \quad \langle \forall i : i \in \textbf{Ord} : & \\
 & & \qquad p' \wedge M.i \overset{U}{\longmapsto} (p' \wedge \langle \exists j : j < i : M.j \rangle) \vee q \rangle \rangle \rangle &
\end{array}
$$

By virtue of these equivalences and by using structural induction over $\mathcal{R}_F$ we can immediately establish that the generalized leads-to relation is well defined:

Theorem 11 (Well-Definedness of $\overset{W}{\longmapsto}$) *For any given program F the equations **(AxEps)**, **(AxAct)**, **(AxSeq)**, **(AxAlt)**, and **(AxStar)** uniquely define a family of relations $\{W : W \in \mathcal{R}_F : (\lambda p.\lambda q.\ p \overset{W}{\longmapsto} q)\}$.*

Moreover, if a triple (W, p, q) satisfies the generalized leads-to relation, it can be shown to do so by a finite number of applications of the above proof rules (due to the finiteness of the structure of W). Therefore every element (W, p, q) in the relation has a finite proof; from now on we write $F \vdash p \overset{W}{\longmapsto} q$ to denote that (W, p, q) satisfies the generalized leads-to relation.

It is also worth mentioning that the use of ordinals is essential in **(AxStar)**: under unconditional fairness it is generally not possible to bound the number of steps required to achieve progress from any particular start state, it can only be asserted that a finite number of steps suffices. Therefore natural numbers as metric are not sufficient; instead all ordinals of cardinality less than or equal to the cardinality of the size of the state space have to be considered.

4.3.2 A Characterization of Provability

The main result of this section is to establish the very close connection between the generalized leads-to relation just defined and the **wltr** predicate transformers of section 4.2. The result is analogous to the one relating the ordinary leads-to relation to the **wlt** predicate transformer in [JKR89, Kna92] and is stated in the following theorem:

Theorem 12 *For any W in $\mathcal{R}_F$ and state predicates p and q in $\mathcal{P}_F$:*

$$[p \Rightarrow \mathbf{wltr}.W.q] \quad \equiv \quad p \stackrel{W}{\longmapsto} q$$

Proof . From theorem 11 the generalized leads-to relation is uniquely defined by the equations **(AxEps)**, **(AxAct)**, **(AxSeq)**, **(AxAlt)**, and **(AxStar)**. It is therefore sufficient to show that the relation **R** defined as

$$\mathbf{R}.W.p.q \quad \equiv \quad [p \Rightarrow \mathbf{wltr}.W.q]$$

solves the same equations. We establish this by induction over the structure of W by observing for all p and q in $\mathcal{P}_F$:

case $W = \varepsilon$:

$$\begin{array}{ll} & \mathbf{R}.\varepsilon.p.q \\ \equiv & \{\text{definition of } \mathbf{R}\} \\ & [p \Rightarrow \mathbf{wltr}.\varepsilon.q] \\ \equiv & \{(\mathbf{wltrEps})\} \\ & [p \Rightarrow q] \\ \equiv & \{(\mathbf{AxEps})\} \\ & p \stackrel{\varepsilon}{\longmapsto} q \end{array}$$

case $W = \alpha$ for some $\alpha \in F.A$:

$$\mathbf{R}.\alpha.p.q \quad \equiv \quad p \stackrel{\alpha}{\longmapsto} q$$

$\equiv$ {definition of $\mathbf{R}$, **(AxAct)**}

$$[p \Rightarrow \mathbf{wltr}.\alpha.q] \quad \equiv \quad \langle \exists p' : [p \Rightarrow p'] : p' \ \mathbf{ensures}_\alpha \ q \rangle$$

which we prove by mutual implication:

$$[p \Rightarrow \mathbf{wltr}.\alpha.q]$$

$\Rightarrow$ {predicate calculus, **(E0)**, $\mathbf{wltr}.\alpha.q$ is witness for p'}

$$\langle \exists p' : [p \Rightarrow p'] : p' \ \mathbf{ensures}_\alpha \ q \rangle$$

$\Rightarrow$ {**(E1)**}

$$\langle \exists p' : [p \Rightarrow p'] : [p' \Rightarrow \mathbf{wltr}.\alpha.q] \rangle$$

$\Rightarrow$ {predicate calculus}

$$[p \Rightarrow \mathbf{wltr}.\alpha.q]$$

case $W = UV$:

$$\mathbf{R}.(UV).p.q$$

$\equiv$ {definition of $\mathbf{R}$}

$$[p \Rightarrow \mathbf{wltr}.(UV).q]$$

$\equiv$ {**(wltrSeq)**}

$$[p \Rightarrow \mathbf{wltr}.U.(\mathbf{wltr}.V.q)]$$

$\equiv$ {monotonicity of $\mathbf{wltr}.U$, predicate calculus,
$\mathbf{wltr}.V.q$ is witness for r}

$$\langle \exists r :: [p \Rightarrow \mathbf{wltr}.U.r] \wedge [r \Rightarrow \mathbf{wltr}.V.q] \rangle$$

$\equiv$ {induction hypothesis, twice}

$$\langle \exists r :: (p \ \overset{U}{\longmapsto} \ r) \wedge (r \ \overset{V}{\longmapsto} \ q) \rangle$$

$\equiv$ {**(AxSeq)**}

$$p \ \overset{(UV)}{\longmapsto} \ q$$

case $W = U + V$:

$$\mathbf{R}.(U + V).p.q$$

$\equiv$ {definition of $\mathbf{R}$}

$$[p \Rightarrow \mathbf{wltr}.(U + V).q]$$

$\equiv$ {**(wltrAlt)**}

$$[p \Rightarrow \mathbf{wltr}.U.q \vee \mathbf{wltr}.V.q]$$

$$\equiv \quad \{\text{predicate calculus, } p \wedge \mathbf{wltr}.U.q \text{ is witness for } r,$$
$$p \wedge \mathbf{wltr}.V.q \text{ is witness for } s\}$$
$$\langle \exists r, s : [r \vee s \equiv p] : [r \Rightarrow \mathbf{wltr}.U.q] \wedge [s \Rightarrow \mathbf{wltr}.V.q] \rangle$$
$$\equiv \quad \{\text{induction hypothesis, twice}\}$$
$$\langle \exists r, s : [r \vee s \equiv p] : r \overset{U}{\longmapsto} q \wedge s \overset{V}{\longmapsto} q \rangle$$
$$\equiv \quad \{\mathbf{(AxAlt)}\}$$
$$p \overset{U+V}{\longmapsto} q$$

case $W = U^*$:

$$\mathbf{R}.U^*.p.q \quad \equiv \quad p \overset{U^*}{\longmapsto} q$$
$$\equiv \quad \{\text{definition of } \mathbf{R}, \mathbf{(AxStar)}\}$$
$$[p \Rightarrow \mathbf{wltr}.U^*.q] \quad \equiv$$
$$\langle \exists p' : [p \Rightarrow p'] : \langle \exists M : M \text{ is metric} :$$
$$\langle \forall i : i \in \mathbf{Ord} :$$
$$p' \wedge M.i \overset{U}{\longmapsto} (p' \wedge \langle \exists j : j < i : M.j \rangle) \vee q \rangle \rangle \rangle$$
$$\equiv \quad \{\text{induction hypothesis}\}$$
$$[p \Rightarrow \mathbf{wltr}.U^*.q] \quad \equiv$$
$$\langle \exists p' : [p \Rightarrow p'] : \langle \exists M : M \text{ is metric} :$$
$$\langle \forall i : i \in \mathbf{Ord} :$$
$$[(p' \wedge M.i) \Rightarrow \mathbf{wltr}.U.((p' \wedge \langle \exists j : j < i : M.j \rangle) \vee q)] \rangle \rangle \rangle$$

In order to proceed with the proof we introduce some notation: first we use the abbreviation $E.p'.M.i$ for the innermost quantification term of the right-hand side of the last formula:

$$E.p'.M.i \equiv [(p' \wedge M.i) \Rightarrow \mathbf{wltr}.U.((p' \wedge \langle \exists j : j < i : M.j \rangle) \vee q)]$$

We also introduce the predicate transformer τ by

$$[\![\tau.X \equiv q \vee \mathbf{wltr}.U.X]\!]$$

Repeated application of τ is defined for any ordinal i in the usual way:

$$[\![\tau^0.X \;\equiv\; X]\!]$$

for step ordinal i: $[\![\tau^i.X \;\equiv\; \tau.(\tau^{i-1}.X)]\!]$

for limit ordinal i: $[\![\tau^i.X \;\equiv\; \langle\exists j : j < i : \tau^j.X\rangle]\!]$

Based on these definitions we state the following properties of τ:

τ is monotonic **(T0)**

$\mathbf{wltr}.U^*.q$ is the strongest solution of $X : [\![\tau.X \equiv X]\!]$ **(T1)**

$[\![\mathbf{wltr}.U^*.q \;\equiv\; \langle\exists i : i \in \mathbf{Ord} : \tau^i.\,\mathrm{false}\rangle]\!]$ **(T2)**

$[\![\tau.\,\mathrm{false} \;\equiv\; q]\!]$ **(T3)**

$\langle\forall i, j : i \le j : [\![\tau^i.\,\mathrm{false} \;\Rightarrow\; \tau^j.\,\mathrm{false}]\!]\rangle$ **(T4)**

(T0) follows from $\mathbf{wltr}.U$ being monotonic, **(T1)** is a restatement of **(wltrStar)**, **(T2)** is a consequence of **(T1)** and the Knaster-Tarski Theorem, **(T3)** follows from **(wltrStrict)**, and **(T4)** finally follows from **(T0)** and **(T3)**.

We restate our proof obligation using the above definitions:

$$[p \Rightarrow \mathbf{wltr}.U^*.q] \quad \equiv \quad (*)$$
$$\langle\exists p' :: [p \Rightarrow p'] \wedge \langle\exists M : M \text{ is metric} : \langle\forall i : i \in \mathbf{Ord} : E.p'.M.i\rangle\rangle\rangle$$

which we prove in the following by mutual implication.

Proof of (*), $\Rightarrow$:

We choose as witnesses for p' and M the following:

$$[p' \;\equiv\; \mathbf{wltr}.U^*.q]$$

$$[M.0 \;\equiv\; \neg\,\mathbf{wltr}.U^*.q]$$

$$\langle\forall i : i > 0 : [M.i \;\equiv\; \tau^i.\,\mathrm{false} \;\wedge\; \neg\langle\exists j : j < i : \tau^j.\,\mathrm{false}\rangle]\rangle$$

In appendix D.1 we show that M is indeed a metric, and furthermore satisfies the following property for all ordinals i:

$[\langle\exists j : 0 < j \le i : M.j\rangle \;\equiv\; \tau^i.\,\mathrm{false}]$ **(M0)**

In order to prove the left-to-right implication of **(*)**, it suffices to show that for the choices for p' and M above:

$$\langle \forall i : i \in \mathbf{Ord} : E.p'.M.i \rangle$$

which we establish by transfinite induction over i. For $i = 0$ we observe

$$\begin{array}{ll}
& E.p'.M.0 \\
\equiv & \{\text{definitions of } M.0,\ p' \text{ and } E\} \\
& [\mathbf{wltr}.U^*.q \wedge \neg\, \mathbf{wltr}.U^*.q \ \Rightarrow \mathbf{wltr}.U.q] \\
\equiv & \{\text{predicate calculus}\} \\
& \text{true}
\end{array}$$

For $i > 0$ we have

$$\begin{array}{ll}
& E.p'.M.i \\
\equiv & \{\text{definitions of } M.i,\ p' \text{ and } E\} \\
& [\mathbf{wltr}.U^*.q \wedge \tau^i.\text{false} \ \wedge \neg\langle \exists j : j < i : \tau^j.\text{false}\rangle \\
& \Rightarrow \quad \mathbf{wltr}.U.((\mathbf{wltr}.U^*.q \wedge \langle \exists j : j < i : M.j\rangle) \vee q)] \\
\equiv & \{\text{from } \mathbf{(T2)}\text{: } [\tau^i.\text{false} \Rightarrow \mathbf{wltr}.U^*.q]\} \\
& [\tau^i.\text{false} \ \wedge \neg\langle \exists j : j < i : \tau^j.\text{false}\rangle \\
& \Rightarrow \quad \mathbf{wltr}.U.((\mathbf{wltr}.U^*.q \wedge \langle \exists j : j < i : M.j\rangle) \vee q)]
\end{array}$$

For a limit ordinal i the antecedent of the last formula is false, hence this last proof obligation is trivially satisfied.

For a step ordinal i we observe

$$\begin{array}{ll}
& [\tau^i.\text{false} \ \wedge \neg\langle \exists j : j < i : \tau^j.\text{false}\rangle \\
& \Rightarrow \quad \mathbf{wltr}.U.((\mathbf{wltr}.U^*.q \wedge \langle \exists j : j < i : M.j\rangle) \vee q)] \\
\Leftarrow & \{\text{left-hand side weakening, splitting the range}\} \\
& [\tau^i.\text{false} \Rightarrow \mathbf{wltr}.U.((\mathbf{wltr}.U^*.q \wedge (M.0 \vee \langle \exists j : 0 < j < i : M.j\rangle)) \vee q)] \\
\equiv & \{\text{definition of } M.0\text{, predicate calculus, } i \text{ is step ordinal}\} \\
& [\tau^i.\text{false} \Rightarrow \mathbf{wltr}.U.((\mathbf{wltr}.U^*.q \wedge \langle \exists j : 0 < j \leq i-1 : M.j\rangle) \vee q)]
\end{array}$$

$\equiv$ {**(M0)**}

$[\tau^i.\text{false} \Rightarrow \textbf{wltr}.U.((\textbf{wltr}.U^*.q \wedge \tau^{i-1}.\text{false}) \vee q)]$

$\equiv$ {**(T2)**, predicate calculus}

$[\tau^i.\text{false} \Rightarrow \textbf{wltr}.U.(\tau^{i-1}.\text{false} \ \vee q)]$

$\Leftarrow$ {monotonicity of $\textbf{wltr}.U$, twice, predicate calculus}

$[\tau^i.\text{false} \Rightarrow \textbf{wltr}.U.(\tau^{i-1}.\text{false}) \ \vee \ \textbf{wltr}.U.q]$

$\Leftarrow$ {**(wrltWeaken)**, predicate calculus}

$[\tau^i.\text{false} \Rightarrow \textbf{wltr}.U.(\tau^{i-1}.\text{false}) \ \vee \ q]$

$\equiv$ {definition of τ, predicate calculus}

true

Proof of **(*)**, $\Leftarrow$:

Let us abbreviate with B the following equation in p':

$$p' : ([p \Rightarrow p'] \ \wedge \ \langle \exists M : M \text{ is metric} : \langle \forall i : i \in \textbf{Ord} : E.p'.M.i \rangle \rangle)$$

In order to establish the right-to-left implication of **(*)**, it clearly suffices to show that any solution p' of B also satisfies also satisfies

$$[p' \Rightarrow \textbf{wltr}.U^*.q]$$

i.e., that $\textbf{wltr}.U^*.q$ is the weakest solution of B. We therefore observe for any p' solving B, where M is some corresponding witness metric:

$[p' \Rightarrow \textbf{wltr}.U^*.q]$

$\equiv$ {**(MetricExh)**, **(T2)** and **(T4)**}

$[p' \wedge \langle \exists i : i \in \textbf{Ord} : M.i \rangle \Rightarrow \langle \exists i : i \in \textbf{Ord} : \tau^{i+2}.\text{false} \rangle]$

$\Leftarrow$ {predicate calculus}

$\langle \forall i : i \in \textbf{Ord} : [p' \wedge M.i \Rightarrow \tau^{i+2}.\text{false}] \rangle$

We establish this last proof obligation by transfinite induction over i. For $i = 0$ we observe:

$p' \wedge M.0$

$$
\begin{aligned}
\Rightarrow\quad & \{p' \text{ solves } B, \text{ definition of } E\} \\
& \mathbf{wltr}.U.q \\
\Rightarrow\quad & \{\text{definition of } \tau\} \\
& \tau.q \\
\equiv\quad & \{\mathbf{(T3)}\} \\
& \tau^2.\text{false}
\end{aligned}
$$

For $i > 0$ we have:

$$
\begin{aligned}
& p' \wedge M.i \\
\Rightarrow\quad & \{p' \text{ solves } B, \text{ definition of } E\} \\
& \mathbf{wltr}.U.((p' \wedge \langle \exists j : j < i : M.j \rangle) \vee q) \\
\equiv\quad & \{\text{predicate calculus}\} \\
& \mathbf{wltr}.U.(\langle \exists j : j < i : p' \wedge M.j \rangle \vee q) \\
\Rightarrow\quad & \{\text{induction hypothesis, } \mathbf{wltr}.U \text{ is monotonic}\} \\
& \mathbf{wltr}.U.(\langle \exists j : j < i : \tau^{j+2}.\text{false} \rangle \vee q)
\end{aligned}
$$

For any limit ordinal i we observe

$$
\begin{aligned}
& \mathbf{wltr}.U.(\langle \exists j : j < i : \tau^{j+2}.\text{false} \rangle \vee q) \\
\Rightarrow\quad & \{i \text{ is limit ordinal}\} \\
& \mathbf{wltr}.U.(\langle \exists j : j < i : \tau^{j}.\text{false} \rangle \vee q) \\
\equiv\quad & \{\text{definition of } \tau^i \text{ for limit ordinal } i, \mathbf{(T3)}\} \\
& \mathbf{wltr}.U.(\tau^i.\text{false} \ \vee \tau.\text{false}) \\
\Rightarrow\quad & \{\mathbf{(T4)}, \text{ predicate calculus}\} \\
& \mathbf{wltr}.U.(\tau^i.\text{false}) \vee q \\
\Rightarrow\quad & \{\text{definition of } \tau, \mathbf{(T4)}\} \\
& \tau^{i+2}.\text{false}
\end{aligned}
$$

and for any step ordinal i we observe

$$
\begin{aligned}
& \mathbf{wltr}.U.(\langle \exists j : j < i : \tau^{j+2}.\text{false} \rangle \vee q) \\
\equiv\quad & \{i \text{ is step ordinal}\} \\
& \mathbf{wltr}.U.(\langle \exists j : j \le i-1 : \tau^{j+2}.\text{false} \rangle \vee q) \\
\equiv\quad & \{\mathbf{(T4)}, \mathbf{(T3)}\}
\end{aligned}
$$

$$
\begin{array}{ll}
& \mathbf{wltr}.U.(\tau^{i+1}.\,\text{false} \ \vee\ \tau.\,\text{false}) \\
\Rightarrow & \{\mathbf{(T4)},\ \text{predicate calculus}\} \\
& \mathbf{wltr}.U.(\tau^{i+1}.\,\text{false}) \vee q \\
\equiv & \{\text{definition of } \tau\} \\
& \tau^{i+2}.\,\text{false}
\end{array}
$$

which concludes the proof.

End of Proof.

As an immediate consequence of the above theorem we observe for all regular expressions W in $\mathcal{R}_F$ and all state predicates p and q on $\mathcal{P}_F$:

$$
\begin{array}{l}
\mathbf{wltr}.W.q \ \stackrel{W}{\longmapsto}\ q \\
p \ \stackrel{W}{\longmapsto}\ q \ \Rightarrow\ [p \Rightarrow \mathbf{wltr}.W.q]
\end{array}
$$

which together establish that $\mathbf{wltr}.W.q$ as defined by the equations **(wltrEps)**, **(wltrAct)**, **(wltrSeq)**, **(wltrAlt)**, and **(wltrStar)** is the unique weakest solution of the equation $p : (p \ \stackrel{W}{\longmapsto}\ q)$.

4.3.3 Some Derived Proof Rules for Generalized Leads-To

In this section we exhibit some useful laws that increase our repertoire of proof rules and enable us to establish more easily generalized leads-to properties of programs. In the following let F be any program.

Theorem 13 (Derived Proof Rules) *For any V, W in $\mathcal{R}_F$ and any state predicates p, p', q, q', and b in $\mathcal{P}_F$, for any set S and any mappings $f, g : S \to \mathcal{P}_F$:*

$$
\begin{array}{ll}
[p \Rightarrow q] \quad \Rightarrow \quad (p \ \stackrel{W}{\longmapsto}\ q) & \textbf{\textit{(Imply)}} \\
{[p' \Rightarrow p]} \ \wedge\ (p \ \stackrel{W}{\longmapsto}\ q) \quad \Rightarrow \quad (p' \ \stackrel{W}{\longmapsto}\ q) & \textbf{\textit{(LhsStr)}} \\
(p \ \stackrel{W}{\longmapsto}\ q) \ \wedge\ [q \Rightarrow q'] \quad \Rightarrow \quad (p \ \stackrel{W}{\longmapsto}\ q') & \textbf{\textit{(RhsWeak)}} \\
\langle \forall m : m \in S : f.m \ \stackrel{W}{\longmapsto}\ g.m \rangle \quad \Rightarrow & \textbf{\textit{(GenDisj)}} \\
\qquad (\langle \exists m : m \in S : f.m \rangle \ \stackrel{W}{\longmapsto}\ \langle \exists m : m \in S : g.m \rangle) &
\end{array}
$$

$$(p \overset{W}{\longmapsto} \text{false}) \quad \Rightarrow \quad [\neg p] \qquad \textit{\textbf{(Impossible)}}$$

$$(p \overset{V}{\longmapsto} q \vee b) \ \wedge \ (b \overset{W}{\longmapsto} r) \quad \Rightarrow \quad (p \overset{VW}{\longmapsto} q \vee r) \qquad \textit{\textbf{(Cancel)}}$$

(Imply) states that the generalized leads-to relation as a binary relation over predicates is weaker than implication. By **(LhsStr)** and **(RhsWeak)** the generalized leads-to relation is weakening in its right and strengthening in its left argument. Finally,**(GenDisj)**, **(Impossible)** and **(Cancel)** are generalizations of the general disjunction, impossibility and cancellation theorems of the ordinary leads-to relation (cf. [CM88]).

Proof of Theorem 13. All proof rules are established by using the fundamental connection between the generalized progress relation and the **wltr** predicate transformers (theorem 12) and properties of **wltr**:

For **(Imply)** we observe for all W, p and q:

$$\begin{array}{ll}
 & p \overset{W}{\longmapsto} q \\
\equiv & \{\text{theorem } 12\} \\
 & [p \Rightarrow \mathbf{wltr}.W.q] \\
\Leftarrow & \{\textbf{(wltrWeaken)}, \text{ predicate calculus}\} \\
 & [p \Rightarrow q]
\end{array}$$

We prove **(LhsStr)** and **(RhsWeak)** simultaneously by observing for all W, p, p', q and q':

$$\begin{array}{ll}
 & p' \overset{W}{\longmapsto} q' \\
\equiv & \{\text{theorem } 12\} \\
 & [p' \Rightarrow \mathbf{wltr}.W.q'] \\
\Leftarrow & \{\text{antecedent } [p' \Rightarrow p], \text{ predicate calculus}\} \\
 & [p \Rightarrow \mathbf{wltr}.W.q'] \\
\Leftarrow & \{\text{antecedent } [q \Rightarrow q'], \textbf{(wltrMon)}, \text{ predicate calculus}\} \\
 & [p \Rightarrow \mathbf{wltr}.W.q] \\
\equiv & \{\text{theorem } 12\} \\
 & p \overset{W}{\longmapsto} q
\end{array}$$

For **(GenDisj)** we observe for all W, S, f and g, where all quantifications are over elements in S:

$$
\begin{array}{cl}
& \langle \forall m :: f.m \overset{W}{\longmapsto} g.m \rangle \\
\equiv & \{\text{theorem } 12\} \\
& \langle \forall m :: [f.m \Rightarrow \mathbf{wltr}.W.(g.m)] \rangle \\
\Rightarrow & \{\text{predicate calculus}\} \\
& \langle \forall m :: [f.m \Rightarrow \langle \exists m :: \mathbf{wltr}.W.(g.m) \rangle] \rangle \\
\Rightarrow & \{\text{predicate calculus, } \mathbf{(wltrMon)}\} \\
& [\langle \exists m :: f.m \rangle \Rightarrow \mathbf{wltr}.W.\langle \exists m :: g.m \rangle] \\
\equiv & \{\text{theorem } 12\} \\
& \langle \exists m :: f.m \rangle \overset{W}{\longmapsto} \langle \exists m :: g.m \rangle
\end{array}
$$

For **(Impossible)** we observe for all W and p:

$$
\begin{array}{cl}
& p \overset{W}{\longmapsto} \text{false} \\
\equiv & \{\text{theorem } 12\} \\
& [p \Rightarrow \mathbf{wltr}.W.\text{false}] \\
\equiv & \{\mathbf{(wltrStrict)}\} \\
& [p \Rightarrow \text{false}] \\
\equiv & \{\text{predicate calculus}\} \\
& [\neg p]
\end{array}
$$

Finally, for **(Cancel)** we observe for all W, p, q, r, and b:

$$
\begin{array}{cl}
& b \overset{W}{\longmapsto} r \\
\equiv & \{\text{theorem 12, predicate calculus}\} \\
& [b \Rightarrow \mathbf{wltr}.W.r] \\
\Rightarrow & \{\text{predicate calculus}\} \\
& [q \vee b \;\Rightarrow\; q \vee \mathbf{wltr}.W.r] \\
\Rightarrow & \{\mathbf{(wltrWeaken)}, \mathbf{(wltrMon)}\} \\
& [q \vee b \;\Rightarrow\; \mathbf{wltr}.W.(q \vee r)] \\
\Rightarrow & \{\mathbf{(wltrMon)}, \mathbf{(wltrSeq)}\} \\
& [\mathbf{wltr}.V.(q \vee b) \Rightarrow \mathbf{wltr}.(VW).(q \vee r)]
\end{array}
$$

$$\Rightarrow \quad \{\text{from antecedent: } [p \Rightarrow \mathbf{wltr}.V.(q \vee b)]\}$$
$$[p \;\Rightarrow\; \mathbf{wltr}.(VW).(q \vee r)]$$
$$\equiv \quad \{\text{theorem 12}\}$$
$$p \overset{VW}{\longmapsto} q \vee r$$

End of Proof.

We conclude our listing of properties of the generalized leads-to relation by relating it to the ordinary leads-to relation of UNITY logic. The soundness and completeness result given below is an immediate corollary of theorems 9 and 12 and the following connection between the ordinary leads-to relation and the **wlt** predicate transformer (cf. [JKR89]):

$$[p \Rightarrow \mathbf{wlt}.q] \quad \equiv \quad p \mapsto q$$

Put together we obtain the following

Corollary 14 *For any W in $\mathcal{R}_F$, and state predicates p and q in $\mathcal{P}_F$:*

$$(p \overset{W}{\longmapsto} q) \quad \Rightarrow \quad (p \mapsto q) \qquad \textit{\textbf{(Sound)}}$$
$$(p \mapsto q) \quad \Rightarrow \quad \langle \exists W : W \in \mathcal{R}_F : p \overset{W}{\longmapsto} q \rangle \qquad \textit{\textbf{(Compl)}}$$

In summary, we see that proving $p \overset{W}{\longmapsto} q$ for some W establishes the ordinary $p \mapsto q$ as well, while conversely any ordinary progress property can be proved as a generalized one.

4.4 Progress and Regular Expressions

After having established properties of the predicate transformers $\mathbf{wltr}.W$ for each W in $\mathcal{R}_F$ in section 4.2, we now turn to the task of investigating the structure of the family of such predicate transformers, i.e., to the task of determining the relationship of predicate transformers $\mathbf{wltr}.U$ and $\mathbf{wltr}.V$ for different regular expressions U and V in $\mathcal{R}_F$.

It is our goal to design an equational theory for such predicate transformers that allows us to relate different regular expressions that may be used as hints in the verification of progress properties. Based on such a theory we are able to design methodologies for effectively using regular expressions in progress proofs, for instance by replacing certain regular expressions by others that allow for a more efficient mechanical verification, or by experimenting with different regular expressions in order to check different hypothesis about design knowledge, to gain a clearer understanding of how progress is achieved, and to help in program and specification debugging.

We will meet the goal of designing an equational theory for $\mathcal{R}_F$ by introducing the notion of a *progress algebra* which captures the essence of the algebraic structure of the **wltr** family of predicate transformers. We could have restricted ourselves to analyzing this family directly, but the introduction of a special algebra for doing so provides a layer of abstraction that makes it possible to separate algebraic issues from the details of the fixpoint characterizations of the predicate transformers, and to compare the proposed algebraic structure to other familiar structures like Kleene algebras [Koz90].

The characterization of the algebraic structure of **wltr** is done in two parts: first, in section 4.4.1, we define the notion of progress algebras and establish a few properties of such algebras. Then, in section 4.4.2, we demonstrate that the family **wltr** of predicate transformers can be considered a progress algebra, which establishes the algebraic characterization of **wltr**.

4.4.1 Progress Algebras

So far we have regarded $\mathcal{R}_F$ as a free algebra generated by the actions in $F.A$ and by the operators ε, $\cdot$ (sequencing), $+$ (alternation), and * (repetition). In the following we define a coarser algebraic structure that we call a *progress algebra* by presenting a list of equalities and equational implications which define a congruence relation on regular expressions and thereby an equivalence class structure.

For a given program F we refer to the resulting algebraic structure as $\mathcal{R}_F$ from now on and call it the *progress algebra* for program F. First we define the equational Horn theory for $\mathcal{R}_F$, then we show that $\mathcal{R}_F$ bears many similarities to the well known Kleene algebras and to the algebra of regular events. In the following we use the familiar formalism and terminology of Kleene algebras [Koz90] where appropriate.

We start with the definition of progress algebra in which the binary relation $\leq$ (pronounced *subsumed by*) is defined by $U \leq V \;\equiv\; U + V = V$.

Definition 5 (Progress Algebra) *A progress algebra $\mathcal{K}$ is the free algebra with binary operations $\cdot$ and $+$, unary operation *, and constant ε satisfying the following equations and equational implications for all U, V, and W in $\mathcal{K}$:*

$$U + (V + W) \;=\; (U + V) + W \qquad \textbf{\textit{(PrAlg0)}}$$

$$U + V \;=\; V + U \qquad \textbf{\textit{(PrAlg1)}}$$

$$W + W \;=\; W \qquad \textbf{\textit{(PrAlg2)}}$$

$$U(VW) \;=\; (UV)W \qquad \textbf{\textit{(PrAlg3)}}$$

$$\varepsilon W \;=\; W \qquad \textbf{\textit{(PrAlg4)}}$$

$$W\varepsilon \;=\; W \qquad \textbf{\textit{(PrAlg5)}}$$

$$UV + UW \;\leq\; U(V + W) \qquad \textbf{\textit{(PrAlg6)}}$$

$$(U + V)W \;=\; UW + VW \qquad \textbf{\textit{(PrAlg7)}}$$

$$\varepsilon \;\leq\; W \qquad \textbf{\textit{(PrAlg8)}}$$

$$\varepsilon + WW^* \;\leq\; W^* \qquad \textbf{\textit{(PrAlg9)}}$$

$$\varepsilon + W^*W \;\leq\; W^* \qquad \textbf{\textit{(PrAlg10)}}$$

$$UW \;\leq\; W \quad \Rightarrow \quad U^*W \;\leq\; W \qquad \textbf{\textit{(PrAlg11)}}$$

$$WU \;\leq\; W \quad \Rightarrow \quad WU^* \;\leq\; W \qquad \textbf{\textit{(PrAlg12)}}$$

*A progress algebra satisfying **(PrAlg11)** but not necessarily **(PrAlg12)** is called a* right-handed *progress algebra, and a progress algebra satisfying **(PrAlg12)** but not* *necessarily* ***(PrAlg11)*** *is called a* left-handed *progress algebra.*

Axioms **(PrAlg0)**, **(PrAlg1)** and **(PrAlg2)** characterize the $+$ operator as associative, commutative, and idempotent. The $\cdot$ operator is associative by **(PrAlg3)** and has ε as a unit by **(PrAlg4)** and **(PrAlg5)**. Axioms **(PrAlg6)** and **(PrAlg7)** define how $\cdot$ and $+$ interact, namely that $\cdot$ distributes over $+$ on the right, but not quite on the left. The next axiom **(PrAlg8)** identifies ε as a minimum element. Finally, the properties of the * operator are characterized by axioms **(PrAlg9)** through **(PrAlg12)**.

The motivation for defining an algebraic structure by the above equations and equational implications is twofold: first, we want to stay as close as possible to the axioms of Kleene algebras ([Koz90]), which define a very important and familiar structure that arises in many different areas of computer science including automata theory and program semantics. Doing so allows us to reuse some theorems of Kleene algebras and regular language theory. Second, we want to capture the equational structure of the family **wltr** of predicate transformers as an algebra. This makes it possible to characterize the essence of the structure of the **wltr** predicate transformers in an abstract way.

Comparing progress algebras with Kleene algebras we notice three major differences:

1. Progress algebras lack the equivalent of the $\emptyset$ constant of Kleene algebras. One could consider introducing such a constant by defining $[\![\mathbf{wltr}.\emptyset.q \equiv \text{false}]\!]$, which would actually satisfy the Kleene axioms referring to $\emptyset$. Since such a regular expression does not have a counterpart in either the operational model or the deductive system, we omit it from further consideration.

2. A progress algebra does not have to satisfy the left distributivity of $\cdot$ over $+$. Only the weaker inequality **(PrAlg6)** is required instead.

3. On the other hand any progress algebra satisfies an additional axiom not present in Kleene algebras **(PrAlg8)**.

The first two differences account for the fact, that progress algebras do not have a ring structure. Whereas the omission of a unit element for $+$ is somewhat arbitrary, the lack of left distributivity of $\cdot$ over $+$ is essential. This lack is a well known property of process algebras [Mil89], which, however, are substantially different due to their lack of any law corresponding to **(PrAlg8)**.

In the following we list some important properties of progress algebras that can be derived from the above axioms. The proofs can be found in appendix D.2. First we have the following properties of the subsumption relation $\leq$:

Lemma 15 *In any left-handed or right-handed progress algebra $\mathcal{K}$, the subsumption relation $\leq$ defined by $U \leq V \equiv U + V = V$ is a partial order. Moreover the sequencing, alternation, and repetition operators are monotonic with respect to $\leq$, i.e., for all U, V, U' and V' in $\mathcal{K}$ with $U \leq V$ and $U' \leq V'$:*

$$UU' \leq VV' \qquad \textbf{\textit{(PrAlgSeq)}}$$

$$U + U' \leq V + V' \qquad \textbf{\textit{(PrAlgAlt)}}$$

$$U^* \leq V^* \qquad \textbf{\textit{(PrAlgStar)}}$$

In spite of the differences from Kleene algebras many theorems of Kleene algebras also hold for progress algebras as stated in the following lemma:

Lemma 16 *In any left-handed or right-handed progress algebra $\mathcal{K}$ for all U, V, and W in $\mathcal{K}$ the following laws hold:*

$$\varepsilon + WW^* = W^* \qquad \textbf{\textit{(PrAlg13)}}$$

$$\varepsilon + W^*W = W^* \qquad \textbf{\textit{(PrAlg14)}}$$

$$W^*W^* = W^* \qquad \textbf{\textit{(PrAlg15)}}$$

$$(W^*)^* = W^* \qquad \textbf{\textit{(PrAlg16)}}$$

Furthermore, for right-handed $\mathcal{K}$ and all U, V, and W in $\mathcal{K}$:

$$V + UW \leq W \;\Rightarrow\; U^*V \leq W \qquad \textbf{\textit{(PrAlg17)}}$$

and for left-handed $\mathcal{K}$ and all U, V, and W in $\mathcal{K}$:

$$V + WU \leq W \;\Rightarrow\; VU^* \leq W \qquad \textbf{\textit{(PrAlg18)}}$$

(PrAlg13) through **(PrAlg16)** correspond to well-known properties of the * operator of Kleene algebras. Furthermore, as in Kleene algebras [Pra88], **(PrAlg17)** is equivalent to **(PrAlg11)**, and **(PrAlg18)** is equivalent to **(PrAlg12)**.

Finally there are some interesting properties of progress algebras that are not generally true in Kleene algebras:

Lemma 17 *In any left-handed or right-handed progress algebra $\mathcal{K}$, for all $n \in \mathbf{N}$ with $n > 0$ and sequences W in $\mathbf{Z}_n \to \mathcal{K}$, for all permutations π of $\mathbf{Z}_n$, and for all U and V in $\mathcal{K}$ the following laws hold:*

$$UV \;\leq\; \varepsilon \quad\Rightarrow\quad U \;=\; \varepsilon \qquad \textbf{\textit{(PrAlg19)}}$$

$$UV \;\leq\; \varepsilon \quad\Rightarrow\quad V \;=\; \varepsilon \qquad \textbf{\textit{(PrAlg20)}}$$

$$U + V \;\leq\; \varepsilon \quad\Rightarrow\quad U \;=\; \varepsilon \qquad \textbf{\textit{(PrAlg21)}}$$

$$U^* \;\leq\; \varepsilon \quad\equiv\quad U \;=\; \varepsilon \qquad \textbf{\textit{(PrAlg22)}}$$

$$UU^* \;=\; U^* \qquad \textbf{\textit{(PrAlg23)}}$$

$$U^*U \;=\; U^* \qquad \textbf{\textit{(PrAlg24)}}$$

$$\langle +U : W : U \rangle \;\leq\; \langle \cdot U : W \circ \pi : U \rangle \qquad \textbf{\textit{(PrAlg25)}}$$

$$\langle +U : W : U \rangle^* \;=\; \langle \cdot U : W \circ \pi : U \rangle^* \qquad \textbf{\textit{(PrAlg26)}}$$

(PrAlg19) through **(PrAlg22)** characterize ε as an irreducible element, **(PrAlg23)** and **(PrAlg24)** show that arbitrary and positive repetitions are equivalent. **(PrAlg25)** states that the alternation of regular expressions is subsumed by any permuted sequencing of those expressions, and **(PrAlg26)** asserts that the repetitions of an alternation of regular expressions and of any permuted sequencing of those expressions are the same.

In order to show that the equational theory of progress algebras is consistent and to illustrate the connection between Kleene algebras and progress algebras, consider $\mathbf{Reg}_\Sigma$, the *algebra of regular events* over the non-empty alphabet Σ, the elements of which are regular languages over Σ. This is an important instance of a Kleene algebra that we relate to a progress algebra in the following.

Clearly, $\mathbf{Reg}_\Sigma$ is not a progress algebra, since **(PrAlg8)** is not satisfied even if W is restricted to denote only non-empty languages: for any α in Σ, $\varepsilon \leq \alpha$ does not hold, since $\mathcal{L}.(\varepsilon + \alpha) = \{\varepsilon, \alpha\} \neq \{\alpha\} = \mathcal{L}.(\alpha)$. Conversely, a progress algebra can certainly not be embedded in a Kleene algebra because of the lack of equality in **(PrAlg6)**. However, if we interpret $\leq$ as the subsumption order on strings (cf. section 2.1.6), and define equality by $U = V \;\equiv\; U \leq V \wedge V \leq U$, it is easy to check that the resulting structure meets all progress algebra axioms (see appendix D.2.4 for details).

4.4.2 $\mathcal{R}_F$ as Progress Algebra

In the previous section we have introduced the notion of progress algebras and have exhibited many properties of them. It is our goal now to show that the family **wltr** of predicate transformers can be regarded as a progress algebra. This allows us to characterize the algebraic structure $\mathcal{R}_F$ of the **wltr** predicate transformers for any program F.

In order to show that **wltr** is a progress algebra, we have to define the equational theory of **wltr**, to relate the operators $\cdot$, $+$, *, and the constant ε of $\mathcal{R}_F$ to operations on predicate transformers, and finally to show that the equations and equational implications defining progress algebras are met by **wltr**.

The equational theory and the algebraic structure of **wltr** are defined as expected: any W in $\mathcal{R}_F$ denotes the predicate transformer $\mathbf{wltr}.W$ over $\mathcal{P}_S$; the meaning of the constant ε is given by **(wltrEps)** as the identity transformer; the meaning of the operators $\cdot$, $+$, and * is given by **(wltrSeq)**, **(wltrAlt)**, and **(wltrStar)** as functional composition of predicate transformers, disjunction of predicate transformers, and a least fixpoint construction respectively; the meaning of the basic elements α in $F.A$ is given by **(wltrAct)** as the $\mathbf{wltr}.\alpha$ predicate transformer. Finally, equality of regular expressions over $F.A$ (written as $=_F$) is defined as equivalence of the corresponding predicate transformers, i.e., for all U, V in $\mathcal{R}_F$:

$$U =_F V \quad \equiv \quad [\![\mathbf{wltr}.U \equiv \mathbf{wltr}.V]\!] \ .$$

The induced subsumption relation $\leq_F$ on $\mathcal{R}_F$ is then given by

$$U \leq_F V \quad \equiv \quad U + V =_F V \ .$$

It follows that for all U and V in $\mathcal{R}_F$:

$$\begin{array}{ll} & U \leq_F V \\ \equiv & \{\text{definition of } \leq_F\} \\ & U + V =_F V \\ \equiv & \{\textbf{(wltrAlt)}, \text{ definition of } =_F\} \\ & [\![\mathbf{wltr}.U \vee \mathbf{wltr}.V \ \equiv \ \mathbf{wltr}.V]\!] \\ \equiv & \{\text{predicate calculus}\} \\ & [\![\mathbf{wltr}.U \ \Rightarrow \ \mathbf{wltr}.V]\!] \end{array}$$

In other words, the subsumption relation $\leq_F$ on $\mathcal{R}_F$ is exactly the implication of the corresponding predicate transformers. Based on this interpretation we can characterize the algebraic structure of $\mathcal{R}_F$ as follows:

Theorem 18 *For any program F, the algebra $\mathcal{R}_F$ is a right-handed progress algebra.*

Proof . We need to show that the progress algebra axioms **(PrAlg0)** through **(PrAlg11)** are satisfied by $\mathcal{R}_F$.

(PrAlg0), **(PrAlg1)**, and **(PrAlg2)** follow by virtue of **(wltrAlt)** directly from the associativity, commutativity, and idempotency of disjunction. Similarly,

(PrAlg3) follows by virtue of **(wltrSeq)** from the associativity of functional composition.

From **(wltrEps)** we know that $\mathbf{wltr}.\varepsilon$ is the identity transformer, which is the left and right identity element of functional composition. This establishes (using **(wltrSeq)**) both **(PrAlg4)** and **(PrAlg5)**.

For **(PrAlg6)** we observe for all q in $\mathcal{P}_F$:

$$
\begin{array}{ll}
 & \mathbf{wltr}.U(V+W).q \\
\equiv & \{\textbf{(wltrSeq)}, \textbf{(wltrAlt)}\} \\
 & \mathbf{wltr}.U.(\mathbf{wltr}.V.q \vee \mathbf{wltr}.W.q) \\
\Leftarrow & \{\mathbf{wltr}.U \text{ is monotonic}\} \\
 & \mathbf{wltr}.U.(\mathbf{wltr}.V.q) \vee \mathbf{wltr}.U.(\mathbf{wltr}.W.q) \\
\equiv & \{\textbf{(wltrSeq)}, \textbf{(wltrAlt)}\} \\
 & \mathbf{wltr}.(UV+UW).q
\end{array}
$$

Similarly, for **(PrAlg7)** we observe for all q in $\mathcal{P}_F$:

$$
\begin{array}{ll}
 & \mathbf{wltr}.(U+V)W.q \\
\equiv & \{\textbf{(wltrSeq)}, \textbf{(wltrAlt)}\} \\
 & \mathbf{wltr}.U.(\mathbf{wltr}.W.q) \vee \mathbf{wltr}.V.(\mathbf{wltr}.W.q) \\
\equiv & \{\textbf{(wltrSeq)}, \textbf{(wltrAlt)}\} \\
 & \mathbf{wltr}.(UW+VW).q
\end{array}
$$

(PrAlg8) follows directly from **(wltrEps)** and **(wltrWeaken)**. For **(PrAlg9)** we see by virtue of **(wltrStar)** that $\mathbf{wltr}.W^*.q$ is a solution of the equation $X : [\![q \vee \mathbf{wltr}.W.X]\!]$, i.e., it satisfies

$$[\![\mathbf{wltr}.W^*.q \;\equiv\; q \vee \mathbf{wltr}.W.(\mathbf{wltr}.W^*.q)]\!]. \tag{\textbf{P0}}$$

With this we observe for all q in $\mathcal{P}_F$:

$$
\begin{array}{ll}
 & \mathbf{wltr}.(\varepsilon+WW^*).q \\
\equiv & \{\textbf{(wltrAlt)}, \textbf{(wltrEps)}, \textbf{(wltrSeq)}\} \\
 & q \vee \mathbf{wltr}.W.(\mathbf{wltr}.W^*.q)
\end{array}
$$

$$\begin{array}{ll} \equiv & \{(\mathbf{P0})\} \\ & \mathbf{wltr}.W^*.q \end{array}$$

The proof of **(PrAlg10)** is slightly more difficult. We first observe for all q in $\mathcal{P}_F$:

$$\begin{array}{ll} & \mathbf{wltr}.(\varepsilon + W^*W).q \\ \equiv & \{(\mathbf{wltrAlt}),\ (\mathbf{wltrEps})\} \\ & q \vee \mathbf{wltr}.W^*W.q \\ \equiv & \{[\![q \Rightarrow \mathbf{wltr}.W^*W.q]\!] \text{ from } (\mathbf{wltrWeaken})\} \\ & \mathbf{wltr}.W^*W.q \\ \equiv & \{(\mathbf{wltrSeq})\} \\ & \mathbf{wltr}.W^*.(\mathbf{wltr}.W.q) \end{array}$$

thus leaving us with the proof obligation

$$[\![\mathbf{wltr}.W^*.(\mathbf{wltr}.W.q) \ \equiv\ \mathbf{wltr}.W^*.q]\!], \qquad (\mathbf{P1})$$

which we prove by mutual implication: the implication from right to left is an immediate consequence of **(wltrWeaken)** and the monotonicity of $\mathbf{wltr}.W^*$. For the other direction we introduce two predicate transformers f and g, that characterize, by virtue of **(wltrStar)**, the left-hand and right-hand side of **(P1)** respectively:

$$[\![f.X \ \equiv\ q \vee \mathbf{wltr}.W.X]\!]$$
$$[\![g.X \ \equiv\ \mathbf{wltr}.W.q \vee \mathbf{wltr}.W.X]\!]$$

Clearly, both f and g are monotonic. Furthermore $\mathbf{wltr}.W^*.q$ is the strongest fixpoint of f, and $\mathbf{wltr}.W^*.(\mathbf{wltr}.W.q)$ is the strongest fixpoint of g. Since both sides of **(P1)** are characterized as strongest fixpoints of some predicate transformers, it seems reasonable to attempt to conduct the proof by finding an *intermediate* predicate characterized as fixpoint of some other monotonic predicate transformer h:

$$[\![\mathbf{wltr}.W^*.(\mathbf{wltr}.W.q) \ \Rightarrow\ \mathbf{wltr}.W^*.q]\!]$$

$$\Leftarrow \quad \{\textbf{(wltrStar)}, \text{ properties of suitable } h\}$$
$$[\![\langle \mu Z :: g.Z\rangle \;\Rightarrow\; \langle \mu Z :: h.Z\rangle]\!] \;\wedge\; [\![\langle \mu Z :: h.Z\rangle \;\Rightarrow\; \langle \mu Z :: f.Z\rangle]\!]$$

For h to satisfy the first conjunct it is sufficient, by theorem 3, to require that $[\![g \Rightarrow h]\!]$. Since $\langle \mu Z :: h.Z\rangle$ is the strongest fixpoint of h, the second conjunct is satisfied by h if $\langle \mu Z :: f.Z\rangle$ is a fixpoint of h. Formally we hence require:

$$[\![g \Rightarrow h]\!]$$
$$[\![h.\langle \mu Z :: f.Z\rangle \;\equiv \langle \mu Z :: f.Z\rangle]\!].$$

Any iterated composition of f satisfies the second condition. Unfortunately $[\![g \Leftarrow f]\!]$ holds, but since f is weakening, we try $f \circ f$ for h by observe for all X in $\mathcal{P}_F$:

$$
\begin{array}{ll}
 & g.X \\
\equiv & \{\text{definition of } g\} \\
 & \mathbf{wltr}.W.q \vee \mathbf{wltr}.W.X \\
\equiv & \{\textbf{(wltrWeaken)}, \text{ predicate calculus}\} \\
 & q \vee \mathbf{wltr}.W.q \vee \mathbf{wltr}.W.X \\
\Rightarrow & \{\mathbf{wltr}.W \text{ is monotonic}\} \\
 & q \vee \mathbf{wltr}.W.(q \vee X) \\
\Rightarrow & \{\mathbf{wltr}.W \text{ is monotonic, } \textbf{(wltrWeaken)}\} \\
 & q \vee \mathbf{wltr}.W.(q \vee \mathbf{wltr}.W.X) \\
\equiv & \{\text{definition of } f, \text{ twice}\} \\
 & f.(f.X)
\end{array}
$$

Hence, choosing $f \circ f$, which is certainly monotonic, for h completes the proof of **(PrAlg10)**.

We finish the proof of theorem 18 by observing that the antecedent of **(PrAlg11)** is

$$[\![\mathbf{wltr}.U.(\mathbf{wltr}.W.q) \;\Rightarrow\; \mathbf{wltr}.W.q]\!], \qquad \textbf{(P2)}$$

whereas the conclusion has the form

$$[\![\mathbf{wltr}.U^*.(\mathbf{wltr}.W.q) \;\Rightarrow\; \mathbf{wltr}.W.q]\!].$$

Since $\mathbf{wltr}.U^*.(\mathbf{wltr}.W.q)$ is, by **(wltrStar)**, the strongest solution of the equation $X : [\![X \;\equiv\; \mathbf{wltr}.W.q \vee \mathbf{wltr}.U.X]\!]$, we have

$$\langle \forall X :: [\![X \;\equiv\; \mathbf{wltr}.W.q \vee \mathbf{wltr}.U.X]\!] \;\Rightarrow\; [\![\mathbf{wltr}.U^*.(\mathbf{wltr}.W.q) \Rightarrow X]\!]\rangle. \qquad \textbf{(P3)}$$

We conclude the proof of **(PrAlg11)** by observing for all q in $\mathcal{P}_F$:

$$\begin{array}{ll}
 & [\![\mathbf{wltr}.U^*.(\mathbf{wltr}.W.q) \;\Rightarrow\; \mathbf{wltr}.W.q]\!] \\
\Leftarrow & \{\textbf{(P3)} \text{ with } X := \mathbf{wltr}.W.q\} \\
 & [\![\mathbf{wltr}.W.q \;\equiv\; \mathbf{wltr}.W.q \vee \mathbf{wltr}.U.(\mathbf{wltr}.W.q)]\!] \\
\equiv & \{\text{predicate calculus, } \textbf{(P2)}\} \\
 & \text{true}
\end{array}$$

End of Proof.

Two remarks about the axioms **(PrAlg6)** and **(PrAlg12)** are in order. First, we note that **(PrAlg6)** cannot be strengthened to equality. By examining the proof for **(PrAlg6)** above we see that for equality we need to prove

$$[\![\mathbf{wltr}.U.(\mathbf{wltr}.V.q \vee \mathbf{wltr}.W.q) \;\equiv\; \mathbf{wltr}.U.(\mathbf{wltr}.V.q) \vee \mathbf{wltr}.U.(\mathbf{wltr}.W.q)]\!]$$

Clearly, we cannot expect this to hold in general, because from lemma 8 we know that $\mathbf{wltr}.U$ is not finitely disjunctive. In fact, the program *FiniteDisjunctivity* of section 4.2.2 serves as a counter example: with $U := \alpha$, $V := \beta$, $W := \gamma$, and $q := (n = 4)$, we have $[\![\mathbf{wltr}.\beta.(n = 4) \equiv n = 2 \vee n = 4]\!]$, $[\![\mathbf{wltr}.\gamma.(n = 4) \equiv n = 3 \vee n = 4]\!]$ and hence by virtue of the calculations in section 4.2.2

$$[\![\mathbf{wltr}.\alpha.(\mathbf{wltr}.\beta.(n = 4) \vee \mathbf{wltr}.\gamma.(n = 4)) \;\equiv\; \text{true}]\!]$$

whereas

$$[\![\mathbf{wltr}.\alpha.(\mathbf{wltr}.\beta.(n=4)) \vee \mathbf{wltr}.\alpha.(\mathbf{wltr}.\gamma.(n=4)) \;\equiv\; n \neq 0]\!]$$

Next, we show that $\mathcal{R}_F$ is not left-handed, i.e., that **(PrAlg12)** is not satisfied. An attempt at a proof of **(PrAlg12)** by transfinite induction over the ordinals up to the ordinal i for which $[\![\mathbf{wltr}.U^*.q \;\equiv\; \tau^i.\text{false}]\!]$, where $[\![\tau.X \equiv q \vee \mathbf{wltr}.U.X]\!]$, fails because the inductive step for limit ordinals seems to require $\mathbf{wltr}.W$ to be or-continuous, which it is not in general by theorem 8.

In the following we construct a counterexample from the failed proof. Such a counterexample can be derived from two regular expressions U and V satisfying

$$\neg[\![\mathbf{wltr}.U^* \Rightarrow \langle \exists i : i < \omega : \mathbf{wltr}.U^i \rangle]\!], \qquad \textbf{(O0)}$$

i.e., $\mathbf{wltr}.U^*$ is not stronger or equal to the ω - disjunction of all $\mathbf{wltr}.U^i$, but also

$$[\![\mathbf{wltr}.V \equiv \langle \exists i : i < \omega : \mathbf{wltr}.U^i \rangle]\!], \qquad \textbf{(O1)}$$

i.e., the ω-disjunction of all $\mathbf{wltr}.U^i$ is expressible[5] in $\mathcal{R}_F$.

We show that such U and V constitute a counterexample by establishing that the antecedent of **(PrAlg12)**, $[\![\mathbf{wltr}.VU \;\Rightarrow\; \mathbf{wltr}.V]\!]$, holds, but that the conclusion $[\![\mathbf{wltr}.VU^* \Rightarrow \mathbf{wltr}.V]\!]$ is not satisfied. For the antecedent we have for all q in $\mathcal{P}_F$

$$\begin{array}{ll}
 & \mathbf{wltr}.VU.q \\
\equiv & \{\textbf{(wltrSeq)}, \textbf{(O1)}\} \\
 & \langle \exists i : i < \omega : \mathbf{wltr}.U^i.(\mathbf{wltr}.U.q) \rangle \\
\equiv & \{\textbf{(wltrWeaken)}, \text{predicate calculus}\} \\
 & \langle \exists i : i < \omega : \mathbf{wltr}.U^i.q \rangle \\
\equiv & \{\textbf{(O1)}\} \\
 & \mathbf{wltr}.V.q
\end{array}$$

A straightforward induction using **(P0)**, **(wltrWeaken)**, and **(wltrSeq)** allows us

[5] We call a predicate transformer τ expressible in $\mathcal{R}_F$ if and only if there is a regular expression V in $\mathcal{R}_F$ such that $[\![\mathbf{wltr}.V \equiv \tau]\!]$.

to establish that $[\![\mathbf{wltr}.U^* \equiv \mathbf{wltr}.U^iU^*]\!]$ for all i in $\mathbf{N}$. For the conclusion we can, therefore, observe for all q in $\mathcal{P}_F$

$$
\begin{aligned}
& \mathbf{wltr}.VU^*.q \\
\equiv\quad & \{\textbf{(wltrSeq)}, \textbf{(O1)}\} \\
& \langle \exists i : i < \omega : \mathbf{wltr}.U^i.(\mathbf{wltr}.U^*.q)\rangle \\
\equiv\quad & \{\textbf{(wltrSeq)}, \text{observation above}\} \\
& \langle \exists i : i < \omega : \mathbf{wltr}.U^*.q\rangle \\
\equiv\quad & \{\text{predicate calculus}\} \\
& \mathbf{wltr}.U^*.q
\end{aligned}
$$

which by **(O0)** and **(O1)** does not imply $\mathbf{wltr}.V.q$.

The following example due to Cohen [Coh96] exhibits such regular expressions U and V, thereby establishing that $\mathcal{R}_F$ is not left-handed. Consider the program *LeftHanded* given by

program *LeftHanded*

  declare

    var n : *natural*

    var b : *boolean*

  assign

$[\alpha]$	$n := n+1$	**if** $\neg b$			
$[\beta]$	$n, b := n-1, \text{true}$	**if** $b \wedge n > 0$	$\sim$	n, true	**if** $\neg b \vee n = 0$
$[\gamma]$	$n, b := 0, \text{true}$	**if** b	$\sim$	n, true	**if**$\neg b$

end

For this program we observe that properties **(O0)** and **(O1)** are satisfied for $U := \beta$ and $V := \gamma$. It can be checked easily that $[\![\mathbf{wltr}.\beta^i.(n = 0) \equiv (n = 0) \vee (b \wedge (n \leq i))]\!]$ for any i in $\mathbf{N}$, $[\![\langle \exists i : i < \omega : \mathbf{wltr}.\beta^i.(n = 0)\rangle \equiv n = 0 \vee b]\!]$, and $[\![\mathbf{wltr}.\gamma.(n = 0) \equiv n = 0 \vee b]\!]$, but that $[\![\mathbf{wltr}.\beta^*.(n = 0) \equiv \text{true}]\!]$.

We conclude our exploration of the algebraic structure of $\mathcal{R}_F$ by combining our results into the main theorem about relating the subsumption relation on progress algebras to implication of the corresponding **wltr** predicate transformers.

From the characterization of $\leq_F$ we recall that that for all U and V in $\mathcal{R}_F$

$$U \leq_F V \;\equiv\; [\![\mathbf{wltr}\,.U \Rightarrow \mathbf{wltr}\,.V]\!]$$

Since, by theorem 18, the family **wltr** of predicate transformers has the structure of a progress algebra, we also know that

$$U \leq V \;\Rightarrow\; U \leq_F V.$$

where the first subsumption is the relation provable in progress algebras, and the second one is the semantic subsumption of elements of $\mathcal{R}_F$. Combining these results we obtain the following:

Theorem 19 *For any program F,* **wltr** *is monotonic with respect to $\leq$ in its first argument, i.e., for all U and V in $\mathcal{R}_F$:*

$$U \leq V \;\Rightarrow\; [\![\mathbf{wltr}\,.U \Rightarrow \mathbf{wltr}\,.V]\!]$$

An immediate consequence of the above theorem and of theorem 12 is the following corollary, which relates the subsumption of regular expressions to the generalized leads-to relation:

Corollary 20 *For any program F and regular expressions U and V in $\mathcal{R}_F$ with $U \leq V$:*

$$(p \overset{U}{\longmapsto} q) \quad\Rightarrow\quad (p \overset{V}{\longmapsto} q).$$

We conclude this section with a brief discussion of the question of how well the **wltr** family of predicate transformers can be characterized algebraically. Theorem 19 establishes that for any given program the implication ordering on the transformers is at least as weak as the subsumption order on the corresponding regular expressions. It is obvious that an exact characterization of the implication order for *individual* programs cannot be achieved by any order on the progress algebra $\mathcal{R}_F$: for a simple program with two (syntactically) identical actions α and β, the predicate transformers $\mathbf{wltr}.\alpha$ and $\mathbf{wltr}.\beta$ are certainly identical, whereas the two regular expressions α and β cannot be related in any progress algebra.

Instead of considering individual programs we might, however, ask about the algebraic structure common to all programs sharing the same action alphabet. More precisely, let A be a finite set of actions, and let $\mathbf{Reg}.A$ be the progress algebra over A. For any regular expressions U and V in $\mathcal{R}.A$ we know that the axiom system for the progress algebra (definition 5) is sound, i.e., that from $U = V$ in $\mathcal{R}.A$ we can conclude $[\![\mathbf{wltr}.U \equiv \mathbf{wltr}.V]\!]$ for any program F with $F.A \supseteq A$. This follows form theorem 19 and the fact that $\mathcal{R}.A$ is a sub-algebra of $\mathcal{R}_F$.

The converse of this observation amounts to the formulation of a *completeness* property of the progress algebra axioms, which can be stated as follows:

$$\langle \forall U, V : U, V \in \mathcal{R}.A : \langle \forall F : F.A \supseteq A : [\![\mathbf{wltr}.U \equiv \mathbf{wltr}.V]\!]\rangle \Rightarrow U = V\rangle$$

It asserts that whenever the predicate transformers corresponding to two regular expressions U and V of some progress algebra $\mathcal{R}.A$ are equivalent *for all programs* F with suitable action set, U and V are provably equivalent in $\mathbf{Reg}.A$. We pose this completeness statement of progress algebras as an open question and remark that a positive answer would establish that the subsumption relation of progress algebras characterizes the *uniform* algebraic structure of the $\mathbf{wltr}$ predicate transformers exactly.

4.5 Progress by Actions

So far we have characterized generalized progress properties by predicate transformers and by a deductive proof system, but are still lacking a formal operational semantics. As mentioned earlier, the main motivation is to provide a link between generalized progress properties characterizing how progress is achieved for a given program, and executions of the program as a sequence of program states and actions, thereby allowing the designer to formulate some operational design knowledge in terms of regular expression hints. In order to allow this link to be exploited during verification, it is important that the operational semantics be as simple and intuitive as possible, and be easily matched with an understanding of possible program executions.

We base our presentation of an operational semantics for generalized progress properties on a certain notion of *games*, inspired by [Dij95], where games are used as a model for the DUALITY calculus. A game takes place between two players, called the *progressor* and the *opponent*, engaging in a series of game rounds. The sequence of the rounds is governed by a strategy chosen by the progressor from a set of possible strategies determined by a regular expression characterizing the game. In a game corresponding to the property $p \overset{W}{\longmapsto} q$, the progressor tries to reach a state satisfying q whenever some state satisfying p has been reached previously, whereas the opponent attempts to prevent this from happening. A program satisfies a property $p \overset{W}{\longmapsto} q$ if the progressor has a winning strategy for reaching q states from p states.

The remainder of this section is organized as follows: in section 4.5.1 we formalize the notions of games and strategies upon which our operational semantics is based. In section 4.5.2 we state our definition of the operational semantics and illustrate it with a few simple examples. Finally, in section 4.5.3 we relate the operational semantics to the deductive system by establishing a soundness and a completeness result.

4.5.1 Games and Strategies

In order to give an operational semantics for the generalized progress properties, we need to formalize the notions of games and strategies for a program F.

A *round* of F is a non-empty, finite run of F, i.e., an element of $(F.A)^+$. If for some α in $F.A$ a round r ends with α, i.e., if $r = x\alpha$ for some $x \in (F.A)^*$, we call r an α-round. A *game* of F is a (finite or infinite) sequence of rounds, i.e., an element of $((F.A)^+)^\infty$. In the following, we denote this set of games of program F by $\mathcal{G}_F$. The concatenation operation $+\!\!+$ on the elements of $\mathcal{G}_F$ is as defined in section 2.1.1.

For every game g we denote by $\bar{g}$ the run obtained by concatenating all rounds of g in order. Formally,

$$\begin{array}{lll} \bar{\langle\rangle} & = & \langle\rangle \\ \bar{g} & = & g.0 +\!\!+ \overline{\mathbf{tail}.g} \qquad\qquad \text{if} \quad |g| > 0 \end{array}$$

Note that $\bar{g}$ is well defined also for infinite g (cf. [Sto81, Bro93]).

Next we will define the notion of a *strategy* of F for regular expressions W in $\mathcal{R}_F$. The *algebra of strategies* $\mathbf{Str}.A.P$ for a given sort A of actions and a sort P of state predicates, is the free sorted algebra generated by the following five constructors, each listed with its respective type:

$$\begin{array}{ll} \mathbf{eps}: & \mathbf{Str}.A.P \\ \mathbf{act}: & A \to \mathbf{Str}.A.P \\ \mathbf{seq}: & (\mathbf{Str}.A.P \times \mathbf{Str}.A.P) \to \mathbf{Str}.A.P \\ \mathbf{alt}: & (P \times \mathbf{Str}.A.P \times \mathbf{Str}.A.P) \to \mathbf{Str}.A.P \\ \mathbf{star}: & (P \times \mathbf{Str}.A.P) \to \mathbf{Str}.A.P \end{array}$$

The mapping $\mathcal{S}$ from $\mathcal{R}_F$ into $\mathbf{Str}.(F.A).\mathcal{P}_F$ associates with each regular expression W the set of possible strategies $\mathcal{S}.W$ of F. The set $\mathcal{S}.W$ is defined inductively over the structure of W as follows. For all α in $F.A$, and all U, V in $\mathcal{R}_F$:

$$\begin{array}{lll} \mathcal{S}.\varepsilon & = & \{\mathbf{eps}\} \\ \mathcal{S}.\alpha & = & \{\mathbf{act}.\alpha\} \\ \mathcal{S}.(UV) & = & \{u, v : u \in \mathcal{S}.U \wedge v \in \mathcal{S}.V : \mathbf{seq}.(u, v)\} \end{array}$$

$$\begin{array}{lll} \mathcal{S}.(U+V) & = & \{t,u,v : t \in \mathcal{P}_F \wedge u \in \mathcal{S}.U \wedge v \in \mathcal{S}.V : \mathbf{alt}\,.(t,u,v)\} \\ \mathcal{S}.U^* & = & \{t,u : t \in \mathcal{P}_F \wedge u \in \mathcal{S}.U : \mathbf{star}\,.(t,u)\} \end{array}$$

A strategy of F for the regular expression W in $\mathcal{R}_F$ is simply an element of $\mathcal{S}.W$. In order to define what it means for a game to satisfy a given strategy, we introduce the relation **sat** over pairs of states and games, and strategies. For a state s, a game g and a strategy w we denote by (s,g) **sat** w that g started in s satisfies w. Formally, **sat** is a binary relation in $(F.S \times \mathcal{G}_F) \times \mathbf{Str}\,.(F.A).\mathcal{P}_F$, defined as the smallest relation satisfying the following conditions for all states s in $F.S$, all games g in $\mathcal{G}_F$, all actions α in $F.A$, all predicates t in $\mathcal{P}_F$, and all strategies u and v in $\mathbf{Str}\,.(F.A).\mathcal{P}_F$:

$$\begin{array}{lll} (s,g)\ \mathbf{sat}\ \ \mathbf{eps} & \text{iff} & |g| = 0 \\ (s,g)\ \mathbf{sat}\ \ \mathbf{act}\,.\alpha & \text{iff} & |g| = 1 \wedge \langle \exists x : x \in (F.A)^* : \bar{g} = x\alpha \rangle \\ (s,g)\ \mathbf{sat}\ \ \mathbf{seq}\,.(u,v) & \text{iff} & \langle \exists e, f : e \mathbin{+\!\!+} f = g : \\ & & \qquad ((s,e)\ \mathbf{sat}\ \ u) \wedge \\ & & \qquad (\mathbf{finite}\,.\bar{e} \Rightarrow ((\bar{e}.s, f)\ \mathbf{sat}\ \ v)) \rangle \\ (s,g)\ \mathbf{sat}\ \ \mathbf{alt}\,.(t,u,v) & \text{iff} & ((s \models t) \Rightarrow ((s,g)\ \mathbf{sat}\ \ u)) \wedge \\ & & ((s \not\models t) \Rightarrow ((s,g)\ \mathbf{sat}\ \ v)) \\ (s,g)\ \mathbf{sat}\ \ \mathbf{star}\,.(t,u) & \text{iff} & ((s \models t) \Rightarrow |g| = 0) \wedge \\ & & ((s \not\models t) \Rightarrow (s,g)\ \mathbf{sat}\ \ \mathbf{seq}\,.(u, \mathbf{star}\,.(t,u))) \end{array}$$

The definition of the algebras **Str** and the sets $\mathcal{S}$ follows very closely the structure of the algebra $\mathcal{R}_F$. The only notable addition to the algebraic structure is the occurrence of the predicate arguments in the constructors **alt** and **star**, which we briefly motivate in the following.

For the regular expression ε and for any action α in $\mathcal{R}_F$ there is only one possible strategy, namely **eps** and **act** $.\alpha$ respectively. Also the strategy for a sequence UV is completely determined by the sub-strategies for U and for V, as a game for UV consists of a game for U followed by a game for V. On the other hand, in alternations and repetitions choices have to be made: in the case of an alternation $U+V$ the progressor can decide whether to play a game for U or a game for V, whereas in the case of a repetition U^*, the progressor can decide after each U game whether to terminate the repetition or to continue with more U games. It is this freedom of choice that is captured by the predicate arguments of the **alt** and **star** strategies: for alternations it is the test predicate that determines which sub-strategy to follow, for repetitions it determines termination.

We illustrate the above definitions with an example: consider the program *UpDown* of the previous section and the regular expression [set][down]*. For the set of strategies for [set][down]* we obtain

$$\mathcal{S}.[\mathrm{set}][\mathrm{down}]^* \;=\; \{t : t \in \mathcal{P}_{UpDown} : \mathbf{seq}.(\mathbf{act}.[\mathrm{set}], \mathbf{star}.(t, \mathbf{act}.[\mathrm{down}]))\}.$$

Choosing for t the predicate $n < 0$, we consider the specific strategy

$$w \;=\; \mathbf{seq}.(\mathbf{act}.[\mathrm{set}], \mathbf{star}.(n < 0, \mathbf{act}.[\mathrm{down}])).$$

We also consider the following games (for all k in $\mathbf{N}$):

$$\begin{aligned} g0 &= \langle\langle[\mathrm{set}], [\mathrm{down}]\rangle\rangle \\ g1 &= \langle\langle[\mathrm{down}], [\mathrm{set}]\rangle\rangle \\ g2.k &= \langle\langle[\mathrm{set}]\rangle, \langle[\mathrm{up}], [\mathrm{down}]\rangle^k\rangle \end{aligned}$$

With these definitions, the following statements about satisfaction of games hold, where s is an arbitrary reachable state of *UpDown*[6]:

$$\begin{aligned} ((s, g0)\ \mathbf{sat}\ w) &\equiv \mathrm{false} \\ ((s, g1)\ \mathbf{sat}\ w) &\equiv (s \models n \leq 0) \\ ((s, g2.k)\ \mathbf{sat}\ w) &\equiv (s \models (k > 0 \wedge n = k - 1) \vee (k = 0 \wedge n < 0)) \end{aligned}$$

Game $g0$ does not satisfy w for any start state because it consists of exactly one round, which is not a [set]-round. Game $g1$, on the other hand, consists of a [set]-

[6] Since the initial predicate of *UpDown* is true, all states of program *UpDown* are reachable.

round followed by zero [down]-rounds. In order to satisfy w it is therefore required that the termination predicate $n < 0$ be true at the end of the [set]-round, which determines the set of possible start states as characterized by $n \leq 0$. Similarly, $g2.k$ consists of one [set]-round and k [down]-rounds, which imposes the requirement that at the end of $g2.k$ the termination predicate should be true, but that it should be false at the end of any intermediate round. The reader can check easily that this requirement results in the above characterization of the possible start states.

4.5.2 An Operational Semantics

Based on the formalization of games and strategies we are now ready to formally define the operational semantics of generalized progress properties; i.e., we define what it means for a program F to be a model of such a property:

Definition 6 (Operational Semantics) *For any program F, state predicates p and q in $\mathcal{P}_F$ and regular expression W in $\mathcal{R}_F$, we say F is a model for the generalized progress property $p \overset{W}{\longmapsto} q$, written $F \models p \overset{W}{\longmapsto} q$, and define it by:*

$$
\begin{aligned}
F \models p \overset{W}{\longmapsto} q \quad & \equiv \\
& \langle \exists w : w \in \mathcal{S}.W : \\
& \qquad \langle \forall s, g : s \textit{ is reachable} : \\
& \qquad\qquad (s \models p) \wedge ((s, g) \textbf{ sat } w) \;\Rightarrow\; (s, \bar{g}) \models q \rangle\rangle
\end{aligned}
$$

According to the definition, F is a model for $p \overset{W}{\longmapsto} q$ if and only if there exists a strategy w with a structure determined by W, such that any game started in a reachable state satisfying p and following the rules of w reaches a state satisfying q after a finite number of actions. The existential quantification in the formula above corresponds to the progressor's ability to choose a particular strategy, whereas the universal quantification reflects the requirement that the strategy be successful regardless of the actions the opponent decides to perform.

We illustrate the operational semantics by considering some special cases for the regular expression W. First, for $W = \alpha$ for some α in $\mathcal{R}_F$, the set $\mathcal{S}.\alpha$ is the singleton $\{\mathbf{act}\,.\alpha\}$. Any game satisfying $\mathbf{act}\,.\alpha$ consists of exactly one α-round. Program F therefore satisfies the generalized progress property $p \overset{\alpha}{\longmapsto} q$ if and only if during any run starting in a reachable state satisfying p and ending with an α action, a state satisfying q is encountered.

Next, we consider the case $W = \alpha^*$ for some α in $\mathcal{R}_F$. Any strategy for α^* is of the form $\mathbf{star}\,.t.(\mathbf{act}\,.\alpha)$ for some state predicate t. Any game satisfying such a strategy is a (finite or infinite) sequence of α-rounds. In order to analyze the operational behavior implied by such a strategy we demonstrate, that $F \models p \overset{\alpha^*}{\longmapsto} q$ if and only if the condition in definition 6 is met by the specific strategy $w = \mathbf{star}\,.q.(\mathbf{act}\,.\alpha)$: clearly, by definition 6, if w satisfies the condition, then $F \models p \overset{\alpha^*}{\longmapsto} q$ holds. Conversely, we need to establish that if the condition is met by any strategy in $\mathcal{S}.\alpha^*$ then it is also met by w. Let v be a strategy establishing $F \models p \overset{\alpha^*}{\longmapsto} q$ and let t be the termination predicate of v. Furthermore let g be any game and s be any state satisfying p. If g is finite then it terminates in a state satisfying q, therefore $(s, \bar{g}) \models q$ holds as required. If g is infinite, we consider the smallest prefix e of g such that $\bar{e}.s \models t$ (or $e = g$ if t is not satisfied after any round of g). Due to the condition for the operational semantics satisfied by v, we have $(s, \bar{e}) \models q$. If e is infinite, we have $e = g$ and therefore $(s, \bar{g}) \models q$. Finally, if e is finite, then $\bar{e}$ is a prefix of $\bar{f}$, from which $(s, \bar{g}) \models q$ follows as well. This establishes that w is a *characterizing strategy* for $p \overset{\alpha^*}{\longmapsto} q$, i.e., a strategy that defines the operational semantics of the given property.

Combining this result with definition 6, we see that program F satisfies the generalized progress property $p \overset{\alpha^*}{\longmapsto} q$ if and only if during any run starting in a reachable state satisfying p and containing infinitely many α actions, a state satisfying q is encountered.

As a last example we have another look at program *UpDown* and demonstrate that it indeed satisfies the property true $\overset{[\text{set}][\text{down}]^*}{\longmapsto}$ $n < 0$. This can be seen by choosing the strategy investigated at the end of section 4.5:

$$w \;=\; \mathbf{seq}\,.(\mathbf{act}\,.[\text{set}], \mathbf{star}\,.(n < 0, \mathbf{act}\,.[\text{down}])).$$

Any game g started in some state s and satisfying w consists of one [set]-round followed by a sequence of [down]-rounds. Since b is true at the end of the [set]-round, and since it remains true once it becomes true, the values of n at the end of each [down]-round form a strictly decreasing sequence. Because termination of the repetition occurs once $n < 0$ holds, we conclude from the well-foundedness of the naturals that g is finite, and that $\bar{g}.s \models n < 0$ holds. Therefore we have $(s, \bar{g}) \models n < 0$ which establishes that $F \models \text{true} \overset{[\text{set}][\text{down}]^*}{\longmapsto} n < 0$.

4.5.3 Soundness and Completeness

In order to relate the deductive system for the generalized leads-to relation of section 4.3 to the operational semantics just defined, we will establish both a soundness and a completeness result. The deductive system is sound with respect to the operational semantics, if and only if any generalized leads-to property proved for a program F is indeed satisfied by F; the deductive system is complete, if and only if any generalized leads-to property that is satisfied by F can actually be proved in the deductive system for F.

To be more precise, we will establish the completeness of the deductive system relative to the expressiveness of the assertion language, in our case the predicate calculus part of the logic. We show that a generalized leads-to property satisfied by a program F can be proved under the assumption that certain sets of states can be characterized by predicates of the assertion language, and that certain predicate transformers and fixpoint operations are expressible in the assertion language as well. This notion of completeness was first investigated by Cook [Coo78], a detailed discussion of some of the issues involved can be found in [Rao95].

We establish the connection between operational semantics and deductive system in the following theorem:

Theorem 21 (Soundness and Completeness) *For any program F and regular expression W in $\mathcal{R}_F$ the deductive system defined for generalized leads-to properties is sound and relatively complete in the sense of Cook:*

$$F \models \quad p \overset{W}{\longmapsto} q \qquad \textit{iff} \qquad F \vdash \quad p \overset{W}{\longmapsto} q$$

In order to prove the above theorem, the notion of a *canonical strategy* $C.W.q$ for a given regular expression W and predicate q is needed. It has the important property that it serves as a witness strategy in the definition of the operational semantics, i.e., whenever $F \vdash p \overset{W}{\longmapsto} q$ holds, we can establish $F \models p \overset{W}{\longmapsto} q$ by virtue of $C.W.q$, and conversely, whenever $F \models p \overset{W}{\longmapsto} q$ can be established by virtue of some strategy, it also can be established by virtue of $C.W.q$. The definition of $C.W.q$ and the proof of the above theorem can be found in appendix D.3.

4.6 Discussion

Our goals for developing the theory of generalized progress have been threefold: such a theory should (i) provide a new way of establishing ordinary progress properties of programs by allowing the user to explicitly characterize how progress is achieved, (ii) make it possible to take advantage of design knowledge in order to more effectively verify programs, and (iii) increase the efficiency of mechanical verification procedures based on the developed theory.

These goals have been achieved by our proposed theory in the following ways: we can (i) prove ordinary progress properties by our generalized ones due to theorem 9; we have (ii) developed an algebraic theory for treating progress hints formally that can be exploited in the verification process; finally, as will be demonstrated in subsequent chapters, we can (iii) take advantage of the new formalism in the form of improved model checking procedures for generalized progress properties.

In summary, the theory of generalized progress makes it possible to incorporate action-based design knowledge into the interactive verification of concurrent systems. This is accomplished by treating hints about how progress is achieved as formal objects (namely as elements of a progress algebra) and by providing a calculus for reasoning about such hints, for relating them to program executions, and for combining them with state-based reasoning methods (such as proving safety properties).

With the theoretical foundations in place, future work on generalized progress will be centered around two questions: the relationship to other formal approaches for verification of progress properties, and the practical application of the new theory to program verification.

On the theoretical side it will be interesting to explore the relationship of our theory to automata-theoretic approaches [Kur94], to deductive systems based on linear temporal logic [M$^+$94], or on the propositional mu-calculus [Koz83, Bra93]. It will be worthwhile to investigate to which extent the ideas of formalizing hints and of incorporating design knowledge at various levels of detail could be exploited by these other approaches.

The application of the our theory to the practical verification of concurrent systems is investigated in the following chapter in the context of finite-state model checking. However, it is important to point out, that there is no inherent restriction of the theory of generalized progress properties to neither finite-state systems, or to model checking as a verification technique. In particular the integration with theorem provers [Gol92, OSR93, GM93] and with infinite state-space model checking approaches, as well as the extension to the compositional structure of UNITY logic [Mis] and to compositional verification in general are of great interest.