UNITY and Büchi automata

Hesselink, W H

Published in:
Formal Aspects of Computing

DOI:
10.1007/s00165-020-00528-x

Publication date:
2021

https://doi.org/10.1007/s00165-020-00528-x

Formal Aspects of Computing (2021) 33: 185–205

Formal Aspects of Computing

UNITY and Büchi automata

Wim H. Hesselink

Bernoulli Institute, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands

Abstract. UNITY is a model for concurrent specifications with a complete logic for proving progress properties of the form "P leads to Q". UNITY is generalized to U-specifications by giving more freedom to specify the steps that are to be taken infinitely often. In particular, these steps can correspond to non-total relations. The generalization keeps the logic sound and complete. The paper exploits the generalization in two ways. Firstly, the logic remains sound when the specification is extended with hypotheses of the form "F leads to G". As the paper shows, this can make the logic incomplete. The generalization is used to show that the logic remains complete, if the added hypotheses "F leads to G" satisfy "F unless G". The main result extends the applicability and completeness of UNITY logic to proofs that a given concurrent program satisfies any given formula of LTL, linear temporal logic, without the next-operator which is omitted because it is sensitive to stuttering. For this purpose, the program, written as a UNITY program, is extended with a number of boolean variables. The proof method relies on implementing the LTL formula, i.e., restricting the specification in such a way that only those runs remain that satisfy the formula. This result is a variation of the classical construction of a Büchi automaton for a given LTL formula that accepts precisely those runs that satisfy the formula.

Keywords: Concurrency, Progress, UNITY, LTL, Büchi automaton

1. Introduction

UNITY [CM88, Mis01] is a formalism to reason about never-terminating concurrent programs or distributed systems. Büchi automata are finite state machines to accept ω-regular languages, see e.g. [GPVW95] and references given there. Both kinds of systems are primarily transition systems. An execution of such a system is an infinite sequence of states in which every pair of subsequent states satisfies the next-state relation. For both kinds of systems, the semantics are given by the set of runs, where a *run* is defined to be an *acceptable* execution. The acceptance criterion will be discussed below.

The important part of UNITY is UNITY logic, a system for deriving assertions of the form $P \mapsto Q$, interpreted as "P leads to Q". UNITY logic is sound and complete for this interpretation. Soundness means that, if $P \mapsto Q$ can be derived for a UNITY program, then, in all runs of it, every state where P holds and Q does not, is followed eventually by a state where Q holds. Completeness of UNITY logic means that, if, in all runs of some program, every state where P holds and Q does not, is followed eventually by a state where Q holds, $P \mapsto Q$ is derivable in UNITY logic for this program.

Correspondence to: Wim H. Hesselink, e-mail: w.h.hesselink@rug.nl

Linear temporal logic (LTL) is a language to express properties of runs. Büchi automata are complete for LTL, in the sense that, for every LTL formula, there is a Büchi automaton that accepts precisely those runs that satisfy the formula.

Coming back to the acceptance of runs, the acceptance criterion for UNITY programs is primarily that all steps are taken infinitely often. A Büchi automaton has a fairness set of states, and the acceptance criterion is that the execution visits the fairness set infinitely often. As will be shown, in both cases the criteria have been generalized in such a way that the two theories meet. At that point, UNITY logic is still sound and complete. One has to be careful, however, with executability. Indeed, a system will be described that, if executable, would solve the Halting Problem!

Two practical differences remain. First, UNITY allows infinite state spaces, e.g., with integers or more complicated data structures, while Büchi automata are finite state machines. Consequently, UNITY programs are usually given in terms of variables, while Büchi automata are described in terms of state diagrams. Secondly, UNITY is insensitive to stuttering to allow abstraction from irrelevant internal steps of components, while for Büchi automata this is not built in, so as not to hamper expressiveness.

In this paper, it is from the side of UNITY that the meeting point is approached. UNITY is generalized to U-specifications because this is possible and preserves soundness and completeness of UNITY logic. Some aspects of the generalization are obvious and useful in practice. Other aspects are useful for the theory, even when they endanger executability.

The first application (of the latter type) concerns leads-to hypotheses. Such hypotheses can be added to UNITY logic in an obvious way, which preserves soundness. It is shown here that it need not preserve completeness. Adding leads-to hypotheses of a special kind preserves completeness. The proof of this uses the generalization to U-specifications.

UNITY logic can only prove assertions of the form $P \mapsto Q$. This seems like a fundamental flaw in expressiveness. The second application of U-specifications enables us, however, to use UNITY logic to prove that a given UNITY program satisfies any given LTL property.

Roughly speaking the method is as follows. Let C be the UNITY program and φ the LTL property one wants to prove. Extend program C with a Büchi automaton to a U-specification E with initialization predicate D in such a way, that $D \mapsto \mathit{false}$ in E means that C has no runs that satisfy $\neg\varphi$. Then use UNITY logic in E to derive $D \mapsto \mathit{false}$. This proves that all runs of C satisfy φ.

The construction of program E from program C and property φ can be split into three parts. First, find an LTL formula α that can be interpreted in C as property φ. Next, construct a Büchi automaton B for the negation $\neg\alpha$. Finally, superimpose B onto C to get E.

The construction of B is the most complicated part, but of course, this construction of an automaton for an LTL formula is known. The constructions in the literature are usually in terms of state diagrams with the explicit aim to make the automaton as small as possible. As one needs to apply UNITY logic to the resulting program E, however, it is important that program E is accessible for analysis. The paper therefore gives a new construction of B in terms of Boolean variables.

Some related research

Around 1990, several formalisms related to UNITY were proposed: action systems [BKS88], specifications [AL91], temporal logic of actions [Lam94]. They all model the execution of the system as a repeated nondeterministic choice between different atomic commands with mild conditions that the choice is made in a "fair" way. The derivation system UNITY logic, however, is unique.

The soundness of UNITY logic is fairly obvious, the completeness was proved in [Kna92, Dij00]. In [Hes13], it was shown that UNITY programs can be generalized to g-unity specifications, called U-specifications here, while retaining soundness and completeness. This step is the starting point for the two extensions of the theory to be presented here.

The problem of adding leads-to hypotheses while preserving completeness of the logic was investigated first by Tsay and Bagrodia [TB95], and Gumm and Zhukov [GZ96]. The present solution is relatively simple because of the generalization of UNITY to U-specifications.

Büchi automata go back to Büchi [Bue60]. The application of temporal logic in concurrency research has been promoted by Manna and Pnueli [MP83], and Lamport [Lam83]. Büchi automata and temporal logic are used together in model checking, see [BK08, Hol04]. Gerth e.a [GPVW95] give a construction of a Büchi automaton for a given LTL formula. This construction is mechanically verified in Isabelle/HOL by Schimpf et al. [SMS09].

Overview

Section 2 introduces the basic material: UNITY, U-specifications, UNITY logic, and the operational semantics. Section 3 treats the addition of leads-to hypotheses to the logic and the cases were the logic becomes incomplete or remains complete. Section 4 introduces linear temporal logic, the concept of implementation, and proves that the validity of an LTL formula on a U-specification can be proved by UNITY logic with an implementation of its negation. This section also contains the treatment of the Halting Problem. The construction of an implementation is done in Sect. 5. Conclusions are drawn in Sect. 6.

Mechanical verification

For the writer's confidence, almost everything in the paper has been mechanically verified with the proof assistant PVS. The dump file is available at [Hes20]. The verification was especially helpful because without intuitive understanding a handwritten formal proof is never completely convincing.

2. UNITY and U-specifications

This section contains a very brief introduction to UNITY programs in Sect. 2.1, followed by a more formal description of U-specifications and UNITY logic in Sect. 2.2. Some linear temporal logic is introduced in Sect. 2.3. Section 2.4 discusses stuttering. The operational semantics are described in Sect. 2.5. Recurring sets and B-specifications are investigated in Sect. 2.6.

2.1. UNITY programs

A UNITY program [CM88, Kna94] consists of state space X, given by a declaration of the available variables and their types, a predicate A that specifies the initial states, followed by an assignment section, which is a set $\mathcal{W}$ of commands R of the form

$$(0) \qquad [] \quad R: \quad B \quad \to \quad S\,.$$

Here B is the guard, the condition under which the assignment S is executed. If B is false, command R does nothing. Command R is identified with the relation that contains a pair (x, y) if and only if execution of R in state x can result in state y. The next-state relation of the program is $N = 1_X \cup \bigcup_{R\in\mathcal{W}} R$, where $1_X = \{(x, x) \mid x \in X\}$. Executing the program means a fair non-deterministic interleaving of the commands $R \in \mathcal{W}$ with optional skip steps of 1_X. The fairness condition is that every command $R \in \mathcal{W}$ is taken infinitely often. Note that each command R is total: if the present state x satisfies B, then S transforms x into the next state; otherwise the next state is x itself.

Example. Consider the UNITY program

$$\begin{array}{l}\textbf{var}\ k : \mathit{int}\\ [] \ R_1 : \ k = 17 \quad \to \quad k := 4\\ [] \ R_2 : \ \mathit{true} \quad \to \quad k := k + 1\end{array}$$

If command R_1 is executed when $k \neq 17$, nothing changes. It follows that the program has two kinds of runs. Those where k cycles infinitely often between 17 and 4, and those where k goes to infinity.

In the next section, non-total relations $R \in \mathcal{W}$ are allowed. If in the above program, R_1 is replaced by the non-total relation $\{(17, 4)\}$, the requirement that command R_1 be taken infinitely often implies that the executions where k goes to infinity are rejected. Then there are no runs that go beyond 17. ♣

Using the initial predicate A and the next-state relation N, the developer of a UNITY program proves a system invariant, say J. This is typically done before any progress of the system is considered, because mistakes in the invariant must be found as soon as possible, and because the progress assertions usually depend on it. Once a satisfactory system invariant has been found, the state space X can be replaced by J, i.e., by the set of the states that satisfy J. This has the effect that J need not be mentioned anymore but can be invoked whenever needed. This is a semantic way of postulating the substitution axiom of Chandy and Misra [CM88, Sect. 3.4].

2.2. U-specifications and UNITY logic

Generalizing UNITY programs, a *U-specification* is defined to be a triple $(X, N, \mathcal{W})$, where N is a reflexive relation on X, and $\mathcal{W}$ is a countable set of relations on X. Relation N is called the step relation or next state relation. The set $\mathcal{W}$ is called the set of fairness relations. The g-unity specifications of [Hes13] are U-specifications with the additional requirement that $R \subseteq N$ for all $R \in \mathcal{W}$. This requirement is eliminated here, at the cost of occasionally replacing R by $R \cap N$.

The generalization has several aspects. Non-trivial steps in N need not be subject to fairness (i.e. not in $\bigcup_{R \in \mathcal{W}} R$), as in [CK97]. This is often necessary. For example, in the case of the dining philosophers, it is important that philosophers are allowed to remain thinking forever. Infinitely many fairness relations R are allowed, and they can be non-deterministic [GP89, Dij95]. The critical generalization is that the fairness relations need not be total [Hes13]. The U-specification is defined to be *total* iff the relation $R \cap N$ is total for all $R \in \mathcal{W}$.

Let $(X, N, \mathcal{W})$ be a U-specification. UNITY is unique in its logic for progress, which is called UNITY logic. This is a derivation system for the operator $\mapsto$, pronounced "leads to". If P and Q are predicates on the state, $P \mapsto Q$ means that, if some run of the system has a state that satisfies $P \wedge \neg Q$ then it has a later state that satisfies Q. The validity of this interpretation is discussed in Sect. 2.5 below.

Recall that, for a relation R and a postcondition Q, the weakest precondition is

$$\mathbf{wp}.R.Q = \{x \mid \forall\ y : (x, y) \in R \ \Rightarrow\ y \in Q\}\ .$$

Before defining $\mapsto$, one defines the relations **co**, **unless**, **recurring**, and **ensures** by

$$\begin{array}{l} P\ \mathbf{co}\ Q \quad \equiv \quad P \subseteq \mathbf{wp}.N.Q\ , \\ P\ \mathbf{unless}\ Q \quad \equiv \quad P \wedge \neg Q\ \ \mathbf{co}\ \ P \vee Q\ . \\ \mathbf{recurring}\ P \quad \equiv \quad (P = X) \vee (\exists\ R \in \mathcal{W} : P \in \mathbf{recur}(R \cap N))\ , \\ \quad \text{where}\ \ P \in \mathbf{recur}.R \quad \equiv \quad \neg\, P \subseteq \mathbf{wp}.R.P\ , \\ P\ \mathbf{ensures}\ Q \quad \equiv \quad P\ \mathbf{unless}\ Q\ \wedge\ \mathbf{recurring}\ (P \Rightarrow Q)\ . \end{array}$$

These notions are illustrated in the example below. Every predicate P satisfies P **co** P because N is reflexive. By definition, the total space X is recurring. Therefore, any inclusion $P \subseteq Q$ implies that P **ensures** Q.

Remarks. Misra's book [Mis01] uses **transient** instead of **recurring**, where $\neg P$ is transient iff P is recurring. Recurrence is preferred here because transience tends to introduce a confusing number of negations in the analysis.

If R is not total, $P \in \mathbf{recur}.R$ can hold while there is no pair $(x, y) \in R$ with $x \notin P$ and $y \in P$. The "step" from $\neg P$ to P is then called *miraculous*. In sequential programming, such steps were forbidden by Dijkstra's Law of the Excluded Miracle [Dij76]. Morris [Mor88] and other authors, however, have argued against this law, see [Hes92, Sect. 1.3] for this and further references. Miraculous steps cannot be expected from any implementation, but they are useful in the theory and do not endanger soundness. ♣

The leads-to operator $\mapsto$ is defined [CM88, Mis01] as the least relation $\mapsto$ on predicates P and Q that satisfies the progress rules

$$\begin{array}{ll} \textit{RuleP0:} & P\ \mathbf{ensures}\ Q \quad \Rightarrow \quad P \mapsto Q\ , \\ \textit{RuleP1:} & P \mapsto U\ \wedge\ U \mapsto Q \quad \Rightarrow \quad P \mapsto Q\ , \\ \textit{RuleP2:} & (\forall\ i \in I : P_i \mapsto Q) \quad \Rightarrow \quad (\exists\ i \in I : P_i) \mapsto Q\ . \end{array}$$

Note that $P \subseteq Q$ implies $P \mapsto Q$ because of *RuleP0*.

One says P **ensures** Q **via** R if P **unless** Q holds and $R \in \mathcal{W}$ and $P \Rightarrow Q \in \mathbf{recur}(R \cap N)$.

Example. Consider the UNITY program

var $i, k : int$
$[\!]\ R_1 :\ i \le k+1 \quad \to \quad i := i+1$
$[\!]\ R_2 :\ k \le i+1 \quad \to \quad k := k+1$

As announced in Sect. 2.1, the step relation is $N = 1_X \cup R_1 \cup R_2$, where R_1 and R_2 are the binary relations corresponding to the guarded commands, labelled by R_1, R_2, respectively. The fairness relations are given by $\mathcal{W} = \{R_1, R_2\}$.

In order to prove that i and k grow arbitrarily large, one can proceed as follows. For arbitrary n, one proves that

$$n \le i+k \ \wedge\ i \le k+1 \ \textbf{ensures}\ n+1 \le i+k \ \textbf{via}\ R_1,$$
$$n \le i+k \ \wedge\ k \le i+1 \ \textbf{ensures}\ n+1 \le i+k \ \textbf{via}\ R_2.$$

The definitions of **ensures**, **unless**, and **co** are used here. By *RuleP0* and *RuleP2*, it follows that $n \le i+k \mapsto n+1 \le i+k$. Using *RuleP1* and induction, one then obtains $n \le i+k \mapsto n+m \le i+k$ for all integer n and natural m. After an application of *RuleP2*, one gets $true \mapsto n \le i+k$. This means that $i+k$ becomes arbitrary large. By similar arguments, one can prove that $true \mapsto i \le k+1$, and that $true \mapsto k \le i+1$. ♣

2.3. State sequences and properties

Unless stated otherwise, all sequences in a state space X are infinite sequences beginning at index 0. The set of these sequences is denoted X^ω.

For a set $U \subseteq X$, the set of sequences that start in U is denoted $[\![\ U\ ]\!]$. For a relation $R \subseteq X^2$, the set of the sequences that begin with an R step is denoted $[\![\ R\ ]\!]_2$. One thus has

$$xs \in [\![\ U\ ]\!] \quad \equiv \quad xs_0 \in U\ ,$$
$$xs \in [\![\ R\ ]\!]_2 \quad \equiv \quad (xs_0, xs_1) \in R\ .$$

For a sequence xs and a number $k \in \mathbb{N}$, the kth suffix of xs is defined to be the sequence $xs \mid k$ with $(xs \mid k)_n = xs_{k+n}$ for all $n \in \mathbb{N}$. For a subset $\varphi \subseteq X^\omega$, the sets $\Box\varphi$ (*always* φ), and $\Diamond\varphi$ (*eventually* φ) are defined by

$$xs \in \Box\varphi \quad \equiv \quad \forall\ k : (xs \mid k) \in \varphi\ ,$$
$$xs \in \Diamond\varphi \quad \equiv \quad \exists\ k : (xs \mid k) \in \varphi\ .$$

Writing $\neg\,\varphi$ for the complement of φ in X^ω, it holds that $\Diamond\varphi = \neg\,\Box\neg\,\varphi$.

Remark. Let $U \subseteq X$. Then $\Box\Diamond[\![\ U\ ]\!]$ consists of the sequences that are infinitely often in U. The set $\Diamond\Box[\![\ U\ ]\!]$ is smaller: it is the strict subset of $\Box\Diamond[\![\ U\ ]\!]$ that consists of the sequences for which from some index onward all elements are in U. ♣

Any subset φ of X^ω is called a *property* on X. To distinguish the properties *false* and *true* from the predicates *false* and *true*, the properties are denoted by $\bot$ and $\top$, respectively. The logical connectives $\vee$, $\wedge$, $\Rightarrow$ are used for properties, just as for predicates, with e.g. $(\varphi \Rightarrow \psi) = (\neg\varphi \vee \psi)$.

2.4. Stuttering, and good properties

A sequence ys is said to be a *stuttering* of a sequence xs iff xs can be obtained from ys by replacing some (possibly an infinite number of) finite nonempty constant subsequences of consecutive elements of ys with their first elements. For example, if a, b, c are different states, the infinite sequence $(aabaaacc)^\omega$ is a stuttering of $(abaac)^\omega$.

According to Lamport's *stutter principle* (e.g., [Lam83, AL91]), the semantics of concurrent systems should be insensitive to stuttering. When discussing properties, one should therefore concentrate on *good* properties defined as follows.

A property $\varphi \subseteq X^\omega$ is called *good* iff $xs \in \varphi$ is equivalent to $ys \in \varphi$ for every sequence xs and stuttering ys of xs. All properties constructed in this paper will be good. To avoid distracting proof obligations, however, the term "property" is not restricted to good properties.

Intersections, unions and complements of good properties are good. If φ is a good property, then $\Box\varphi$ and $\Diamond\varphi$ are good. Also, $[\![\ U\ ]\!]$ is a good property for any subset $U \subseteq X$.

For a reflexive relation R, the property $\Box[\![\ R\ ]\!]_2$ is good. It follows that $\Diamond[\![\ R\ ]\!]_2$ is good whenever R is irreflexive. According to the stutter principle, $\Diamond[\![\ R\ ]\!]_2$ should only be used for irreflexive relations R. If R is not necessarily irreflexive, the term $\Diamond[\![\ R\ ]\!]_2$ is therefore replaced by $\Diamond[\![\ R\ ]\!]_+$ where $[\![\ R\ ]\!]_+$ is defined by

$$[\![\ R\ ]\!]_+ = [\![\ R\ ]\!]_2 \cup [\![\ \mathbf{stut}.R\ ]\!] \text{ where } \mathbf{stut}.R = \{x \mid (x,x) \in R\}\ .$$

The index + of $[\![\ \ ]\!]_+$ serves to remind us that the argument is a relation and that stuttering is taken into account. As $\Diamond$ distributes over unions, the set $\Diamond[\![\ R\ ]\!]_+$ is the union of $\Diamond[\![\ R \setminus 1_X\ ]\!]_2$ and $\Diamond[\![\ \mathbf{stut}.R\ ]\!]$. The latter two properties are good because $R \setminus 1_X$ is irreflexive and $\Diamond$ preserves goodness. Therefore $\Diamond[\![\ R\ ]\!]_+$ is good. It captures the intention of $\Diamond[\![\ R\ ]\!]_2$, because $xs \in \Diamond[\![\ R\ ]\!]_+$ holds iff $\Diamond[\![\ R\ ]\!]_2$ contains a stuttering of xs.

2.5. Operational semantics

The operational semantics of the U-specifications of Sect. 2.2, and also of B-specifications (Sect. 2.6), L-specifications (Sect. 3.1), and U*-specifications (Sect. 4.2) that will be introduced later, are all expressed by means of the following general semantical concept of specification.

A *specification* is a triple $K = (X, N, \varphi)$ where X is a set, the state space, N is a reflexive relation on X, and φ is property on X. For a specification K the set of runs is defined by $\mathbf{run}.K = \Box[\![\ N\ ]\!]_2 \cap \varphi$. So, a run is a sequence xs with $(xs_i, xs_{i+1}) \in N$ for all $i \in \mathbb{N}$, that satisfies φ.

Remark. Abadi and Lamport [AL91] define a specification to be a tuple (X, N, A, ψ) where X is a set (the state space), N is a reflexive relation on X, A is a subset of X (of the initial states), and ψ is a good property. The set of behaviours of such an AL-specification is $\Box[\![\ N\ ]\!]_2 \cap [\![\ A\ ]\!] \cap \psi$. An AL-specification induces the specification in our sense by taking $\varphi = [\![\ A\ ]\!] \cap \psi$. Conversely, if one ignores the requirement that ψ is good, a specification in our sense induces an AL-specification by taking $A = X$ and $\psi = \varphi$. The only difference therefore is that goodness is ignored. For shortness, the word behaviour is replaced by run. ♣

A sequence xs with $(xs_i, xs_{i+1}) \in N$ for all $i \in \mathbb{N}$ is called an *execution*. A nonempty finite sequence with the same property is called an *incomplete execution*. Specification K is said to be *machine closed* [AL91] iff every incomplete execution can be extended to a run. Note that an infinite execution need not be a run, because it need not satisfy φ.

Specification K is said to satisfy a property ψ, notation $K \models \psi$, if and only if all its runs satisfy ψ. In other words, one has

$$K \models \psi \quad \equiv \quad \mathbf{run}.K \subseteq \psi\ .$$

In [Hes13], the function **LT** is defined by $\mathbf{LT}.P.Q = \Box([\![\ P\ ]\!] \Rightarrow \Diamond[\![\ Q\ ]\!])$. One says that P **leads-to** Q if $K \models \mathbf{LT}.P.Q$.

The following lemma is a variant of the PSP rule of [Mis01]:

Lemma 1 Let P, Q, A, B be predicates on X. Assume that P **leads-to** Q and that A : **co** : B. Then $P \wedge B$ **leads-to** $(Q \vee \neg A) \wedge B$.

Proof Let xs be a run with $xs_k \in P \wedge B$. One has to prove that $xs_n \in (Q \vee \neg A) \wedge B$ holds for some $n \geq k$.

Assume to the contrary that $xs_n \in \neg((Q \vee \neg A) \wedge B)$ for all $n \geq k$. As $\neg((Q \vee \neg A) \wedge B) = \neg B \vee (A \wedge \neg Q)$, one has $xs_n \in B \ \Rightarrow\ xs_n \in A \wedge \neg Q$ for all $n \geq k$. It follows from the assumption A **co** B that $xs_n \in A$ implies $xs_{n+1} \in B$ for all n. We have $xs_k \in B$. By induction, it follows that $xs_n \in B$ for all $n \geq k$, and hence that $xs_n \in \neg Q$ for all $n \geq k$. This contradicts the assumption P **leads-to** Q. □

Back to U-specifications. The semantics of a U-specification $A = (X, N, \mathcal{W})$ are determined by the *associated* specification $\sigma.A = (X, N, \varphi)$ where $\varphi = \bigcap_{R \in \mathcal{W}} \Box\Diamond[\![\ R\ ]\!]_+$. For every $R \in \mathcal{W}$, the conjunct $\Box\Diamond[\![\ R\ ]\!]_+$ expresses that there are infinitely many indices n such that $(xs_n, xs_{n+1}) \in R$ or $(xs_n, xs_n) \in R$. This is called *impartiality* in [Hes13]. It is a variation of weak fairness. One can use a scheduler [Hes13, Sect. 4.1] to prove

Proposition 2 Let A be a total U-specification. Then the associated specification $\sigma.A$ is machine closed.

One can therefore argue that a U-specification is executable if and only if it is total.

The operational concept of **leads-to** of specification $\sigma.A$ is lifted implicitly to the U-specification A. Then the operational interpretation of $P \mapsto Q$ in Sect. 2 amounts to the assertion that P **leads-to** Q. Indeed, in [JKR89, Hes13], it is proved that

Theorem 3 In every U-specification, P **leads-to** Q if and only if $P \mapsto Q$.

The assertion "if" is soundness of the logic, the "only if" is completeness.

2.6. Recurring sets

The expression for **recur** in Sect. 2.2 can be simplified. It is easy to verify that

(1) $$P \in \mathbf{recur}.R \quad \equiv \quad \forall\, (x, y) \in R : x \in P \;\vee\; y \in P\ .$$

It follows that every set that contains a recurring set is recurring.

Instead of imposing some relations to be impartial, one can choose to impose some predicates to be *recurring*. This is done in the concept of B-specification.

A *B-specification* is defined to be a triple $K = (X, N, \mathcal{V})$ where N is a a reflexive relation on X, and $\mathcal{V}$ is a countable set of sets. The associated specification is $\sigma.K = (X, N, \varphi_R)$ where $\varphi = \bigcap_{V \in \mathcal{V}} \Box\Diamond[\![\ V\ ]\!]$.

Remark. The B of B-specifications refers to Büchi. Indeed, the generalized Büchi automata of [GPVW95, Sect. 3] have the same acceptance condition, and are therefore B-specifications with an initialization and a finite state space. Ordinary Büchi automata, see e.g., [BK08, Hol04], are the special case where $\mathcal{V}$ consists of a single set. ♣

A B-specification $K = (X, N, \mathcal{V})$ is equivalent to the *associated* U-specification $K' = (X, N, \mathcal{W})$ with $\mathcal{W} = \{\rho.V \mid V \in \mathcal{V}\}$, where $\rho.V = \{(x, x) \mid x \in V\}$. The specifications K and K' have the same sets of runs because $[\![\ \rho.V\ ]\!]_+ = [\![\ V\ ]\!]$. Formula (1) implies that $P \in \mathbf{recur}(\rho.V)$ is equivalent to $V \subseteq P$.

The following example shows that not every U-specification is equivalent to an B-specification:

Example. Consider the U-specification K_0 of a ring of size $n \geq 3$:

$$\begin{array}{ll} \mathbf{var}\ k : X = \{0 \ldots n-1\} & \\ [\!] \quad R: & k := (k+1)\ \mathbf{mod}\ n \\ [\!] \quad L: & k := (k-1)\ \mathbf{mod}\ n \end{array}$$

where both commands R and L, to turn right or left, are treated impartially. Let cs be the execution in which command R is always executed and never L. This sequence satisfies $\Box N$, and $\Box\Diamond[\![\ P\ ]\!]$ for every nonempty set P, but it does not satisfy $\Box\Diamond[\![\ L\ ]\!]_+$, and it is not a run of K_0. This proves that U-specification K_0 is not equivalent to any B-specification.

Let K_1 be the B-specification $(X, N, \mathcal{V})$ where N is as in K_0, and $\mathcal{V}$ consists of the sets V that are recurring in K_0. Then K_0 and K_1 have the same relation $\mapsto$, but K_1 has more runs that K_0 (e.g. cs). There exist properties ψ with $K_0 \models \psi$ and $K_1 \not\models \psi$. Indeed, $\Box\Diamond[\![\ L\ ]\!]_+$ is such a property. ♣

3. Adding leads-to hypotheses

Consider an assertion like

(2) Every run in which F leads to G, is such that P leads to Q,

where F, G, P, and Q are predicates on the state. We do not write that $F \mapsto G$ implies $P \mapsto Q$, because this would mean that, if F leads to G for every run of the system, then P leads to Q for every run of the system. A priori, implication (2) is stronger. Implication (2) is formalized by regarding $F \mapsto G$ as a hypothesis that restricts the set of runs of the system. The aim then is to use this hypothesis to prove that $P \mapsto Q$ holds for all (restricted) runs. One may want to add any number of such hypotheses. For example, in a threading system with mutual exclusion, one may want to investigate the hypothesis that every thread q in the entry protocol, will eventually be in the critical section, as formalized in

$$\forall\ q:\ \ q\ \mathbf{in}\ \mathit{entry}\ \mapsto\ q\ \mathbf{in}\ \mathit{CS}\ .$$

The introduction of leads-to hypotheses is formalized by the concept of L-specifications in Sect. 3.1. In Sect. 3.2, it is shown that, in general, the resulting logic is incomplete. Section 3.3 shows that completeness is retained if the L-specification is very moderate. This result is generalized to moderate L-specifications in Sect. 3.4.

3.1. L-specifications

An L-specification extends a U-specification with a set of leads-to hypotheses $\mathcal{L}$. It is defined to be a tuple $K = (X, N, \mathcal{W}, \mathcal{L})$ where $(X, N, \mathcal{W})$ is a U-specification and $\mathcal{L}$ is a countable set of pairs of predicates. The semantics is determined by the associated specification $\sigma.K = (X, N, \varphi)$ given by

$$\varphi \quad = \quad \bigcap_{R \in \mathcal{W}} \Box\Diamond[\![R]\!]_+ \cap \bigcap_{(F,G)\in\mathcal{L}} \mathbf{LT}.F.G \; .$$

The first conjunct imposes impartiality of the members of $\mathcal{W}$. The pairs $(F, G) \in \mathcal{L}$ are called *leads-to hypotheses*. The second conjunct restricts the runs to those that satisfy the leads-to hypotheses.

For an L-specification $(X, N, \mathcal{W}, \mathcal{L})$, the leads-to operator $\mapsto$ is defined as the least relation on predicates P and Q that satisfies the rules *RuleP0*, *RuleP1*, *RuleP2*, as well as

RuleP3: $F \mapsto G$ for every pair $(F, G) \in \mathcal{L}$,
RuleP4: If $P \mapsto Q$ and A **co** B, then $P \wedge B \mapsto (Q \vee \neg A) \wedge B$.

RuleP3 introduces the leads-to hypotheses of $\mathcal{L}$. *RuleP4* is justified by Lemma 1. It follows that the rules *RuleP0*, up to *RuleP4* are sound in the sense that $P \mapsto Q$ implies P **leads-to** Q for every pair of predicates P, Q.

In general, the rules are not complete as is shown in Sect. 3.2 below. In Sect. 3.4, it is proved that the rules are complete, if restricted to so-called moderate L-specifications.

3.2. Incompleteness with leads-to hypotheses

The following argument shows that, in general, the system of the UNITY rules *RuleP0*, ..., *RuleP4* is incomplete. Consider an L-specification with X as the only recurring set, and with four predicates P, Q, R, S, that satisfy $\neg P$ **co** $\neg P$ and the three leads-to hypotheses

$$(3) \qquad \begin{array}{l} P \wedge Q \mapsto P \wedge \neg Q \; , \\ P \wedge \neg Q \mapsto P \wedge Q \; , \\ R \mapsto S \; . \end{array}$$

Every run that satisfies the first two hypotheses satisfies P **co** P, because $\neg P$ is stable and a state that satisfies P needs to toggle Q while remaining inside P. Therefore, the system satisfies $P \wedge R$ **leads-to** $P \wedge S$. This consequence, however, is not derivable by means of the rules *RuleP0*, ..., *RuleP4*.

In order to prove this, one considers a different model of the rules *RuleP0*, ..., *RuleP4* in which this consequence does not hold. This model consists of a state space X with a reflexive next state relation N. Relation **unless** is defined as in Sect. 2.2. The set X is the only recurring set. Therefore, P **ensures** Q is equivalent to $P \subseteq Q$. The relation $\mapsto$ is defined by

$$P \mapsto Q \quad \equiv \quad (\forall\; x \in P : \exists\; y \in Q : (x, y) \in N^*) \; .$$

The rules *RuleP0*, *RuleP1*, *RuleP2* hold trivially. An easy inductive argument shows that *RuleP4* also holds.

Now specialize to $X = \{1, 2, 3\}$. Let N consist of the identity relation together with the three transitions $2 \to 1$, $1 \to 2$, and $2 \to 3$. Take $P = \{1, 2\}$, $Q = R = \{2\}$, and $S = \{3\}$. The safety property $\neg P$ **co** $\neg P$ holds because state 3 has no transitions to 1 or 2. The three hypotheses of (3) precisely correspond to the three transitions. Yet the consequence $P \wedge R$ **leads-to** $P \wedge S$ is false because $P \wedge R = \{2\}$ and $P \wedge S$ is empty.

3.3. Very-moderate completeness

Let an L-specification $(X, N, \mathcal{W}, \mathcal{L})$ be called *very-moderate* if $F = X$ (i.e., the predicate *true*) for every pair $(F, G) \in \mathcal{L}$.

Example. Consider the UNITY program with a single integer variable k, the assignment

$$[] \quad k := k + 1 \,,$$

and the leads-to hypothesis $\mathit{true} \mapsto k = 7$. This is a very-moderate L-specification.

Predicate $A : 7 < k$ satisfies A **co** A. Therefore, *RuleP4* implies $A \mapsto \mathit{false}$. On the other hand, by *RuleP0*, the incrementation of k gives $k = 7 \mapsto A$. Finally, transitivity gives $\mathit{true} \mapsto \mathit{false}$. This means that the L-specification has no runs. Operationally, this is also obvious because in every run k goes beyond 7. It follows that the L-specification is not machine-closed. Therefore, by Proposition 2, the L-specification is not equivalent to any total U-specification. ♣

A very-moderate L-specification K can be transformed into the U-specification in which the leads-to axioms $X \mapsto G$ of K are replaced by making G recurring, i.e., by postulating that $\rho.G = \{(x, x) \mid x \in G\}$ is impartial for every pair $(X, G) \in \mathcal{L}$, see Sect. 2.6. More precisely, the L-specification $K = (X, N, \mathcal{W}, \mathcal{L})$ is transformed into the U-specification $\pi.K = (X, N, \mathcal{W}')$ with $\mathcal{W}' = \mathcal{W} \cup \{\rho.G \mid (X, G) \in \mathcal{L}\}$.

Lemma 4 Let K be a very-moderate L-specification. Then the U-specification $\pi.K$ has the same runs and the same relations $\mapsto$ and **leads-to** as K. For both K and $\pi.K$, the relations $\mapsto$ and **leads-to** are equal.

Proof First, K and $\pi.K$ have the same runs because they have the same step relation N and $\mathbf{LT}.X.G = \Box\Diamond[\![\, \rho.G \,]\!]_+$ for every pair $(X, G) \in \mathcal{L}$.

It follows that $(\textbf{leads-to}_{\pi.K}) = (\textbf{leads-to}_K)$. As $\pi.K$ is a U-specification, plain completeness (Theorem 3) implies $(\mapsto_{\pi.K}) = (\textbf{leads-to}_{\pi.K})$. Soundness of $\mapsto$ for all L-specifications implies that $(\mapsto_K) \subseteq (\textbf{leads-to}_K)$.

It therefore remains to prove that $(\mapsto_{\pi.K}) \subseteq (\mapsto_K)$. We first prove that

$$P \ \textbf{ensures}_{\pi.K} \ Q \quad \Rightarrow \quad P \mapsto_K Q \,. \tag{4}$$

As K and $\pi.K$ have the same step relation N, they have the same **unless** relation. The antecedent of (4) therefore implies $P \ \textbf{unless}_K \ Q$. It also implies that $P \Rightarrow Q \in \textbf{recur}.R$ for some $R \in \mathcal{W}'$. If $R \in \mathcal{W}$, this implies that $P \ \textbf{ensures}_K \ Q$ and hence $P \mapsto_K Q$.

The main case is therefore that $R = \rho.G$ for some pair $(X, G) \in \mathcal{L}$. In this case, $P \Rightarrow Q \in \textbf{recur}.R$ is equivalent to $P \wedge G \subseteq Q$ because of the definition of $\rho.G$. On the other hand, using *RuleP4*, the axiom $X \mapsto_K G$, and $P \wedge \neg Q$ **co** $P \vee Q$ because of P **unless** Q, one obtains

$$X \wedge (P \vee Q) \ \mapsto_K \ (G \vee \neg(P \wedge \neg Q)) \wedge (P \vee Q) \,.$$

This reduces to $P \vee Q \mapsto_K (P \wedge G) \vee Q$. As $P \wedge G \subseteq Q$, this implies formula (4).

Relation $\mapsto_{\pi.K}$ is the least relation between predicates on the state space that satisfies *RuleP0*, *RuleP1*, *RuleP2* using $\textbf{ensures}_{\pi.K}$. As relation $\mapsto_K$ also satifies these rules, it follows that $\mapsto_{\pi.K}$ is contained in $\mapsto_K$. □

3.4. Moderate completeness

An L-specification $(X, N, \mathcal{W}, \mathcal{L})$ is called *moderate* iff every pair $(F, G) \in \mathcal{L}$ satisfies F **unless** G. It follows that every very-moderate L-specification is moderate. It now remains to extend the result of the previous section from very-moderate L-specifications to moderate ones.

The main idea is that any moderate leads-to formula $F \mapsto G$ is equivalent to the very-moderate property $X \mapsto \neg F \vee G$. Something similar holds in the operational semantics. More precisely, in any L-specification K, it holds that

Lemma 5 Let F and G be predicates.
(a) The relation $F \mapsto G$ implies $X \mapsto \neg F \vee G$.
(b) If F **unless** G holds, then relation $X \mapsto \neg F \vee G$ implies $F \mapsto G$.
(c) $\mathbf{LT}.F.G \subseteq \mathbf{LT}.X.(\neg F \vee G)$.
(d) If F **unless** G holds, then $(\mathbf{LT}.X.(\neg F \vee G) \wedge \textbf{run}.K) \subseteq \mathbf{LT}.F.G$.

Proof (a) The assumption $F \mapsto G$ weakens to $F \mapsto \neg F \vee G$. We also have $\neg F \mapsto \neg F \vee G$ because $\neg F$ is a subset. Therefore, *RuleP2* gives $X \mapsto \neg F \vee G$.

(b) We have $F \wedge \neg G$ **co** $F \vee G$ because F **unless** G. *RuleP4* with $X \mapsto \neg F \vee G$ results in:

$$X \wedge (F \vee G) \;\mapsto\; ((\neg F \vee G) \vee \neg(F \wedge \neg G)) \wedge (F \vee G)\,.$$

The lefthand side equals $F \vee G$, and is therefore implied by F. The righthand side reduces to G.

(c) Let $xs \in \mathbf{LT}.F.G$. In order to prove $xs \in \mathbf{LT}.X.(\neg F \vee G)$, assume $xs_n \in X$ for some index n. One has to prove that $xs_k \in \neg F \vee G$ for some $k \geq n$. If $xs_n \notin F$, one can take $k = n$. Otherwise, one uses $xs \in \mathbf{LT}.F.G$ to infer that $xs_k \in G$ for some $k \geq n$.

(d) Let $xs \in \mathbf{LT}.X.(\neg F \vee G) \wedge$ **run**$.K$. In order to prove $xs \in \mathbf{LT}.F.G$, let $xs_n \in F$ for some index n. One has to prove that $xs_k \in G$ for some $k \geq n$. As F unless G holds and xs is a run of K, the sequence xs satisfies $xs_i \in F \;\Rightarrow\; xs_{i+1} \in F \vee G$ for all indices i. It follows that either $xs_k \in G$ for some $k \geq n$, or $xs_k \in F$ for all $k \geq n$. Finally, use $xs \in \mathbf{LT}.X.(\neg F \vee G)$. □

Given a moderate L-specification $K = (X, N, \mathcal{W}, \mathcal{L})$, we construct the very-moderate L-specification $\omega.K = (X, N, \mathcal{W}, \mathcal{L}')$ with $\mathcal{L}' = \{(X, \neg F \vee G) \mid (F, G) \in \mathcal{L}\}$.

The axioms $X \mapsto \neg F \vee G$ of L-specification $\omega.K$ are derivable in K because of Lemma 5 (a). The axioms $F \mapsto G$ of K are derivable in $\omega.K$ because of Lemma 5(b). It follows that L-specifications K and $\omega.K$ have the same relations $(\mapsto_K) = (\mapsto_{\omega.K})$.

One uses Lemma 5(c) and (d) to prove that the L-specifications K and $\omega.K$ have the same runs. This implies that **leads-to**$_K$ = **leads-to**$_{\omega.K}$.

Lemma 6 Let K be a moderate L-specification. Then the L-specification $\omega.K$ has the same runs and the same relations $\mapsto$ and **leads-to** as K.

As $\omega.K$ is very-moderate, Lemma 4 gives $(\mapsto_{\omega.K}) =$ (**leads-to**$_{\omega.K}$). This proves moderate completeness:

Theorem 7 For a moderate L-specification K, the relations $\mapsto_K$ and **leads-to**$_K$ are equal.

4. Validity and LTL formulas

As explained in the Introduction, validity of an LTL formula can be proved by means of an implementation of its negation. The proof of this result requires some heavy definitions, but then it is fairly simple. It is given in Sect. 4.2.

Roughly speaking, an implementation of a property φ on a U-specification A is a U-specification C with an initial predicate D such that every run of A that satisfies φ corresponds to a run of C that starts in D, and vice versa. The formal definition is given in Sect. 4.2.

The construction of implementations is postponed to Sect. 5. To give an impression how it is done, Sect. 4.3 describes an implementation of a property with three temporal operators. As a warning not to expect the impossible, this example is used to "solve" the Halting Problem in Sect. 4.4.

Section 4.5 defines linear temporal logic, LTL, with its operators for always, eventually, release, and until. LTL has two sides: a semantic side with LTL properties and a syntactic side with LTL formulas. The connection is that the formulas are interpreted as properties.

4.1. Action of functions

In this investigation, functional simulations are used, and not relational ones, because functions are simpler than relations, and are good enough for the purpose.

Let f be a function $X \to Y$. This function transfers sequences in X^ω to sequences in Y^ω because a sequence xs in X is a function $\mathbb{N} \to X$, so that the composition $f \circ ys : \mathbb{N} \to Y$ is a sequence in Y. The function transfers a predicate P on Y backwards to the predicate $P \circ f$ on X. It also transfers relations and temporal properties backward. If R is a binary relation on Y, then $f^\bullet R$ is the relation on X that contains the pairs (x, x') with $(f.x, f.x') \in R$. If φ is a property on Y, i.e., a subset of Y^ω, then $f^\bullet\varphi = \{xs \in X^\omega \mid f \circ xs \in \varphi\}$ is the induced property on X.

4.2. U*-specifications

A U*-specification extends a U-specification with a temporal property φ. It is defined to be a tuple $K = (X, N, \mathcal{W}, \varphi)$ such that $(X, N, \mathcal{W})$ is a U-specification and φ is a property on X. The associated specification is $\sigma.K = (X, N, \psi)$ where $\psi = (\bigcap_{R\in\mathcal{W}} \Box\Diamond[\![R]\!]_+) \wedge \varphi$.

If K is a specification of some kind, its constituents are denoted by X_K, N_K, $\mathcal{W}_K$, etc. In the remainder of this section, variables K and L range over arbitrary U*-specifications, and A, C, E range over U-specifications.

Given a U-specification A and a property φ on the state space X_A of A, the U*-specification $A * \varphi$ is defined by $A * \varphi = (X_A, N_A, \mathcal{W}_A, \varphi)$. Of course, every U*-specification K can be written $K = A * \varphi$ with a U-specification A in a unique way. The set of runs of $A * \varphi$ is the conjunction $\mathbf{run}.A \wedge \varphi$. It follows that $A * \varphi \models \psi$ is equivalent to $A \models (\varphi \Rightarrow \psi)$. A U*-specification is called *atomic* iff φ is an atomic property, i.e., of the form $[\![D]\!]$ for some predicate D. The notation $A * \varphi$ is convenient because the aim is to implement φ, i.e., to replace φ by an atomic property.

If $f : X_L \to X_K$ is a function between the state spaces of specifications L and K, it is convenient to speak of a function $f : L \to K$ by abuse of notation. A function $f : L \to K$ is defined to be a *refinement function* iff it satisfies

$$\forall\ ys \in \mathbf{run}.L : f \circ ys \in \mathbf{run}.K\ ,$$

and to be a *corefinement function* iff

$$\forall\ xs \in \mathbf{run}.K : \exists\ ys \in \mathbf{run}.L : f \circ ys = xs\ .$$

It is defined to be a *birefinement function* iff it is both a refinement and a corefinement function.

Remark. If f is a refinement mapping as defined by Abadi and Lamport [AL91], it is a refinement function. The converse implication does not hold.

The concept of birefinement function is not directly related to bisimulation between transition systems, because bisimulation works in branching temporal logic and compares computation trees, while birefinement functions work in linear temporal logic and compares runs. ♣

The specialization from a temporal property φ to an initialization $[\![D]\!]$ is formalized in the following definition. An *implementation* of a U*-specification $A * \varphi$ is defined to be a birefinement function $f : C * [\![D]\!] \to A * \varphi$ where C is a U-specification and D is a predicate on X_C.

Note that $C * [\![D]\!]$ is equivalent to C if $D = \mathit{true}$. The term implementation must be understood in a strict sense. The fact that f is a refinement function means that every run of $C * [\![D]\!]$ induces a run of $A * \varphi$. That it is a corefinement function, adds to this that every run of $A * \varphi$ is represented. In other words, all non-deterministic choices of $A * \varphi$ are still possible for the implementation.

For every U-specification A and every LTL property φ, the U*-specification $A * \varphi$ has an implementation. The proof of this fact is postponed to Theorem 21 in Sect. 5 below. At this point, we can show its relevance. It enables us to use UNITY logic to prove validity of φ, by means of the following result.

Theorem 8 Let $f : C * [\![D]\!] \to A * (\neg\varphi)$ be an implementation. Then $A \models \varphi$ holds if and only if $D \mapsto_C \mathit{false}$.

Proof $A \models \varphi$ means that all runs of A satisfy φ. In other words, it means that A has no runs that satisfy $\neg\varphi$, or equivalently, that the U*-specification $A * (\neg\varphi)$ has no runs. As $f : C * [\![D]\!] \to A * (\neg\varphi)$ is an implementation, this is equivalent to $C * [\![D]\!]$ having no runs. In U-specification C, every suffix of a run is a run. Therefore, $C * [\![D]\!]$ has no runs if and only if D **leads-to**$_C$ false. By plain completeness (Theorem 3), this is equivalent to $D \mapsto_C \mathit{false}$. □

Remark. This result means that, for the claim $A \models \varphi$, an implementation gives an initialized U-specification $C * [\![D]\!]$ such that the validity of the claim can either be proved with UNITY logic applied to C, or refuted by exhibiting a run of $C * [\![D]\!]$. Perhaps such a run can be found by model checking. ♣

4.3. An example

Let C be a U-specification on a state space Y. Let F, G, H, be predicates on Y. Consider the property

$$\varphi \quad = \quad \Diamond[\![H]\!] \ \wedge\ \Box([\![F]\!] \vee \Box[\![G]\!])\ .$$

When applied to φ, the construction of an implementation $f : E * [\![D]\!] \to C * \varphi$ in Sect. 5.3 uses three auxiliary Boolean variables: b_0 for $\Diamond[\![H]\!]$, and b_1 for $\Box[\![G]\!]$, and b_2 for the outer $\Box$. So the new state space is the Cartesian product $Z = Y \times \mathbb{B}^3$, and $f : Z \to Y$ is the natural projection. The step relation of E is given by

$$\begin{array}{ll} & ((y, b_0, b_1, b_2), (y', b_0', b_1', b_2')) \in N_E \\ \equiv & \quad (y, y') \in N_C \ \wedge \ (b_0 \ \Rightarrow \ b_0' \vee H.y') \\ & \wedge \ (b_1 \wedge G.y \ \Rightarrow \ b_1' \wedge G.y') \\ & \wedge \ (b_2 \wedge (F.y \vee b_1 \wedge G.y) \ \Rightarrow \ b_2' \wedge (F.y' \vee b_1' \wedge G.y')) \, . \end{array}$$

Note that primes refer to the next state. The initial predicate D is

$$D \quad = \quad (b_0 \vee H.y) \wedge b_2 \wedge (F.y \vee b_1 \wedge G.y) \, .$$

The fairness set $\mathcal{W}_E$ consists of the relations inherited from $\mathcal{W}_C$, augmented with $\rho.(\neg b_0)$ to make $\neg b_0$ recurring. This means that

$$\mathcal{W}_E \quad = \quad \{\rho.(\neg b_0)\} \vee \{f^\bullet R \mid R \in \mathcal{W}_C\} \, .$$

The reader is invited to try and prove that $f : E * [\![D]\!] \to C * \varphi$ is birefinement function, i.e. maps runs of $E * [\![D]\!]$ to runs of $C * \varphi$ and that every run of $C * \varphi$ is obtained in this way.

4.4. Solving the halting problem!?

Let $A = (X, N, \mathcal{W})$ be a total U-specification. Let P be a stable predicate on the state space X. One can regard reaching P as termination. In other words, the Halting Problem is a special case of the problem to decide whether predicate P is reached or not.

To solve this problem, specification A is extended with a Boolean variable m to a total U-specification C. The Boolean m is a message that P will never be reached. The predicate m is made stable in C and, when it is false, it can become true non-deterministically. This is formalized in the step relation N_1 given by $(m, m') \in N_1 \equiv (m \Rightarrow m')$. The full U-specification becomes $C = (Y, N', \mathcal{W})$ where $Y = X \times \mathbb{B}$ and $((x, m), (x', m')) \in N' \equiv (x, x') \in N \ \wedge \ (m, m') \in N_1$, while the fairness set $\mathcal{W}$ is inherited from $\mathcal{W}$. The first component of a run of C is just a run of A.

The idea that m determines non-termination is expressed in the LTL formula

$$\varphi \quad = \quad \Diamond[\![P \vee m]\!] \ \wedge \ \Box([\![m]\!] \Rightarrow \Box[\![\neg P]\!]) \, .$$

A run of C satisfies φ if and only if it always reaches a state where $P \ \vee \ m$ holds, and if m ever holds, then henceforth P is false. In other words, any run in φ halts after finitely many steps or sends a message m indicating that it will never halt.

This property φ is a case of the LTL formula of Sect. 4.3 with $H = (P \vee m)$, $F = \neg m$, and $G = \neg P$. One can therefore use its implementation $f : E * [\![D]\!] \to C * \varphi$. It is easy to verify that, for every $x \in X$, there exists $z \in D$ with $f.z = (x, \mathit{false})$.

The runs of E that start in D induce runs of C that satisfy φ, and all runs of C that satisfy φ are represented in this way. Assume that every state in D is the starting point of a run of E. Then, if one wants to know whether a computation of A with initial state $x \in X$ terminates, one can submit it to the U*-specification $E * [\![D]\!]$. Choose $y \in D$ with $f.y = (x, \mathit{false})$, extend y to a run of E, execute this run until $P \ \vee \ m$ holds. If P, the computation of A has terminated. If m, it will never terminate. This shows that U*-specification $E * [\![D]\!]$ solves the Halting Problem for A.

If A is undecidable, this is a contradiction. Therefore, the initial state $y \in D$ that was used cannot be extended to a run of E. This implies that E is not machine closed. By Proposition 2, it follows that U-specification E is not total. As U-specification C is total, it follows that the nontotality was introduced by the construction of the implementation. Indeed, the constructions of Sect. 5.6 often give nontotal impartiality relations.

4.5. Linear temporal logic, LTL properties and formulas

Linear temporal logic, LTL, is defined as follows. The unary temporal operators $\Diamond$ and $\Box$ are special cases of the binary operators **U** (*until*) and **R** (*release*) given by

$$\begin{array}{lcl} xs \in \varphi \,\mathbf{U}\, \psi & \equiv & \exists\ k : (xs \mid k) \in \psi \ \wedge\ (\forall\ i : i < k \ \Rightarrow\ (xs \mid i) \in \varphi)\,, \\ xs \in \varphi \,\mathbf{R}\, \psi & \equiv & \forall\ k : (xs \mid k) \in \psi \ \vee\ (\exists\ i : i < k \ \wedge\ (xs \mid i) \in \varphi)\,. \end{array}$$

Indeed, $\Diamond\varphi = \top\,\mathbf{U}\,\varphi$ and $\Box\varphi = \bot\,\mathbf{R}\,\varphi$. The operators **U** and **R** are dual in the sense that $\varphi\,\mathbf{R}\,\psi = \neg(\neg\varphi\,\mathbf{U}\,\neg\psi)$. If φ and ψ are good properties, then $\varphi\,\mathbf{U}\,\psi$ and $\varphi\,\mathbf{R}\,\psi$ are good.

The set of *LTL properties* on a set X is defined inductively as the least set Φ of properties on X such that

$$\begin{array}{l} [\![\, P \,]\!] \in \Phi \ \text{ for every predicate } P \text{ on } X, \\ \varphi \wedge \psi,\ \varphi \vee \psi,\ \varphi\,\mathbf{U}\,\psi,\ \varphi\,\mathbf{R}\,\psi \in \Phi \ \text{ for every pair } \varphi, \psi \in \Phi. \end{array}$$

The properties of the form $[\![\, P \,]\!]$ are called *atomic* properties.

All properties in Φ are good. It holds that $\bot = [\![\, \mathit{false} \,]\!] \in \Phi$ and $\top = [\![\, \mathit{true} \,]\!] \in \Phi$. For every $\varphi \in \Phi$, we have $\neg\varphi \in \Phi$. This is proved by induction, by pushing the negation inward and finally using $\neg\,[\![\, P \,]\!] = [\![\, \neg\, P \,]\!]$. For every $\varphi \in \Phi$, it holds that $\Box\varphi = \bot\,\mathbf{R}\,\varphi \in \Phi$, and $\Diamond\varphi = \top\,\mathbf{U}\,\varphi \in \Phi$.

Remark. Traditionally (e.g. [MP83]), LTL has a next operator **X**, such that $\mathbf{X}\varphi$ means that φ holds after one step. As **X** is sensitive to stuttering, and insensitivity to stuttering is an essential feature of UNITY, this operator is not treated here. ♣

The LTL properties are tied to the state space X via the atomic properties. LTL formulas are introduced to abstract from the state space. They are obtained by replacing the atomic properties by symbols (i) with $i \in \mathbb{N}$. The set of *LTL formulas* is thus defined syntactically as the least set Ψ of formulas such that

$$\begin{array}{l} (i) \in \Psi \ \text{ for every } i \in \mathbb{N}, \\ \alpha \wedge \beta,\ \alpha \vee \beta,\ \alpha\,\mathbf{U}\,\beta,\ \alpha\,\mathbf{R}\,\beta \in \Psi \ \text{ for every pair } \alpha, \beta \in \Psi. \end{array}$$

To interpret an LTL formula α, a state space is needed where the symbols (i) represent boolean variables and the symbols $\wedge$, $\vee$, **U**, **R** have meaning as logical and temporal operators. The state space used here is the set of sets of natural numbers $Z = \mathbb{P}(\mathbb{N})$. For any set (state space) X and any function $f : X \to Z$, the interpretation of α on X via f is defined recursively as the property on X given by

$$\begin{array}{l} [\![\, (i) \,]\!]_f = [\![\, \{x \in X \mid i \in f.x\} \,]\!]\,, \\ [\![\, \beta \oplus \gamma \,]\!]_f = [\![\, \beta \,]\!]_f \oplus [\![\, \gamma \,]\!]_f\,, \end{array}$$

for all i, β, γ, and all operator symbols $\oplus \in \{\wedge, \vee, \mathbf{R}, \mathbf{U}\}$. The index f is omitted if f is the identity function $Z \to Z$.

Lemma 9 (a) For every LTL formula α and every function $f : X \to Z$, it holds that $[\![\, \alpha \,]\!]_f$ is an LTL property on X with $[\![\, \alpha \,]\!]_f = f^{\bullet}[\![\, \alpha \,]\!]$.

(b) For every LTL property φ on X, there is an LTL formula α and a function $f : X \to Z$ with $\varphi = f^{\bullet}[\![\, \alpha \,]\!]$.

Proof (a) This is straightforward by induction on the structure of α.

(b) One first proves that every LTL property φ has a finite expression by means of the atomic building blocks and the operators $\wedge$, $\vee$, **R**, and **U**. Write each atomic building block as $[\![\, P_i \,]\!]$ for some predicate P_i on X, with $i \in \mathbb{N}$. Let α be the LTL formula obtained by replacing each $[\![\, P_i \,]\!]$ by (i). Let I be the set of indices i for which P_i has been defined. Let function $f : X \to Z$ be defined by

$$f.x = \{i \in I \mid x \in P_i\}\,.$$

It follows that $[\![\, (i) \,]\!]_f = [\![\, P_i \,]\!]$ for all $i \in I$. As P_i only occurs in φ when $i \in I$, it follows that $\varphi = [\![\, \alpha \,]\!]_f$. □

Remark. One represents the properties $\top$ and $\bot$ by $[\![\, X \,]\!]$ and $[\![\, \emptyset \,]\!]$, respectively.

5. Constructing an implementation of an LTL property

As announced in Sect. 4.2, for every LTL property on a U-specification A, the U*-specification $A * \varphi$ has an implementation. The present section is devoted to the proof of this and its contruction.

This section contains more manipulation of subsets and boolean functions than elsewhere. Subsets are identified with boolean functions, and we write $P.x$ or $P(x)$ or $x \in P$, all with the same meaning, for predicate (subset) P and state x.

For the construction, it is useful to regard the set $Z = \mathbb{P}(\mathbb{N})$ of Sect. 4.5 as the state space of a trivial U-specification $A_0 = (Z, Z^2, \emptyset)$. The step relation of A_0 is Z^2, which means that every step is possible. There are no fairness requirements. Therefore, every state sequence is a run of A_0.

Consider a U*-specification $A * \varphi$ where φ is an LTL-property on the state space X of A. The starting point of the construction is Lemma 9(b) that yields an LTL formula α and a function $f : X \to Z$ such that $\varphi = f^\bullet[\![\alpha]\!]$. The subsequent construction of an implementation of $A * \varphi$ is split into two parts:

1. Construct an implementation $g : C * [\![D_0]\!] \to A_0 * [\![\alpha]\!]$.
2. Lift the implementation g to an implementation $E * [\![D]\!] \to A * \varphi$.

Part 1 is the difficult part, treated in the Subsects. 5.1 up to 5.7. Part 2 is treated in Sect. 5.8.

5.1. The construction of an automaton for an LTL formula

For every LTL formula there exists a Büchi automaton that accepts precisely those runs that satisfy it, as was shown in the mid eighties [Hol04, p. 141]. As observed in Sect. 2.6, a generalized Büchi automaton is the same as an initialized B-specification on a finite state space. For the construction of such an automaton, one can refer to Gerth et al. [GPVW95]. This construction has been verified mechanically in Isabelle/HOL by Schimpf et al. [SMS09]. To apply this construction in Theorem 8, however, one would have to translate this automaton into a U-specification.

Therefore, in this section a construction is proposed that serves the purpose. The construction gives a B-specification, because this is simpler than a U-specification and is easily translated into a U-specification, see Sect. 2.6. The construction has a complete proof that has been mechanically verified in PVS.

The construction was conceived completely independent of the classical construction, indeed not at all thinking of automata. The design of the construction is focussed on the proof, not on the size of the resulting automaton. The recursive scheme of the construction differs completely from the one used by Gerth et al. [GPVW95]. It is therefore unlikely that knowledge of this classical construction would help to understand the present one.

For easy translation between the B-specification for an LTL formula and the B-specifications for its subformulas, all B-specifications in this section are constructed in a single state space, where countably many fresh boolean variables are available. For lack of better names, the kinds of specifications constructed in this state space are called skeletons, automata, houses, and star-houses.

The state space and the skeletons and automata in it are introduced in Sect. 5.2. Section 5.3 gives the recursive construction of the automaton for any LTL formula. The correctness proof of the construction begins with syntactic matters in Sect. 5.4, followed by semantic matters in Sect. 5.5. In Sect. 5.6, the specific constructions for release (**R**) and until (**U**) are treated. Section 5.7 concludes the correctness proof for the automaton construction.

5.2. Skeletons and automata

As explained in Sect. 4.5, LTL formulas are interpreted in the state space $Z = \mathbb{P}(\mathbb{N})$, that is spanned by the propositional variables. Another sequence of boolean variables are added to make space for the B-specifications to be constructed. The state space used is thus the Cartesian product $W = \mathbb{P}(\mathbb{N}) \times \mathbb{P}(\mathbb{N})$. The elements of W are written as pairs (y, z). The *auxiliary* variables B_n and the *propositional* variables Q_n are the boolean functions on W given by $B_n(y, z) = (n \in y)$ and $Q_n(y, z) = (n \in z)$ for $(y, z) \in W$, respectively. The natural projections on the first and second component are denoted $p_1 : W \to \mathbb{P}(\mathbb{N})$ and $p_2 : W \to Z$.

The first step in the construction of automata is formed by the skeletons, which are defined as follows. A *skeleton* is a tuple $H = (N, \mathcal{V}, F)$ where N is a reflexive relation on the state space W, and $\mathcal{V}$ is a finite set of predicates on W, and F is a finite set of natural numbers. Relation N is called the *step relation*, $\mathcal{V}$ is called the set of *fairness sets*, and F is called the *frame*. This frame will be used later to indicate which auxiliary variables are used.

An *automaton* is a pair (H, D) where H is a skeleton and D is a predicate on W, which may be used as the initial predicate.

A skeleton $H = (N, \mathcal{V}, F)$ is defined to be *extended* by skeleton $H' = (N', \mathcal{V}', F')$, notation $H \sqsubseteq H'$, iff $N \supseteq N'$ and $\mathcal{V} \subseteq \mathcal{V}'$ and $F \subseteq F'$. It is clear that relation $\sqsubseteq$ is a partial order. The smallest skeleton with respect to this order is the trivial skeleton *Triv*, given by $Triv = (W^2, \emptyset, \emptyset)$. This means that in *Triv* every step is allowed and all sequences are runs. The next construction operator for skeletons is joining. The *join* of skeletons $H = (N, \mathcal{V}, F)$ and $H' = (N', \mathcal{V}', G)$ is defined by $H \sqcup H' = (N \cap N', \mathcal{V} \cup \mathcal{V}', F \cup G)$. Indeed, relation $\sqsubseteq$ makes the set of the skeletons into a lattice, and $\sqcup$ is the join of this lattice.

5.3. The construction

First, the temporal operators **R** (release) and **U** (until) are treated. This is done by defining functions that, given a skeleton H, predicates P and Q on W, and starting point $m \in \mathbb{N}$, return an automaton for the properties $[\![P]\!] \mathbf{R} [\![Q]\!]$ and $[\![P]\!] \mathbf{U} [\![Q]\!]$, extending H. The parameter m is a counter used for the creation and numbering of fresh boolean variables. The functions are defined as follows. Assume $H = (N, \mathcal{V}, F)$. Both functions use an auxiliary variable B_m. They are defined by

$$\begin{aligned}
&\mathit{Release}(H, P, Q, m) = ((N \cap N', \mathcal{V}, F \cup \{m\}), B_m \wedge Q) \\
&\quad \text{where, for } x, y \in W, \\
&\quad (x, y) \in N' \quad \equiv \quad (B_m.x \wedge Q.x \wedge \neg P.x \;\Rightarrow\; B_m.y \wedge Q.y) \,. \\
&\mathit{Until}(H, P, Q, m) = ((N \cap N', \mathcal{V} \cup \{\neg B_m\}, F \cup \{m\}), (B_m \wedge P) \vee Q) \\
&\quad \text{where, for } x, y \in W, \\
&\quad (x, y) \in N' \quad \equiv \quad (B_m.x \wedge P.x \;\Rightarrow\; (B_m.y \wedge P.y) \vee Q.y) \,.
\end{aligned}$$

In either case, the step relation is restricted, the frame is extended with m, and the initial condition is expressed in terms of B_m, P, and Q. In the case of *Until*, a fairness set is added.

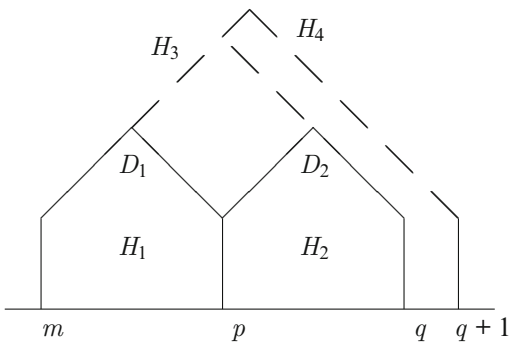
The automaton for an LTL formula α is constructed by a function *Auto*, defined by recursion on the structure of α. This function has two arguments, the LTL formula α, and a natural number m that indicates the leftmost point of the frame for the construction. This m plays the same role as above. The function returns a triple. The first two components form an automaton, while the third component is a natural number, the righthand limit of the frame used.

If the LTL formula α is a leaf (i), the definition refers to the propositional variable Q_i. It is

$$\mathit{Auto}((i), m) = (\mathit{Triv}, Q_i, m) \,.$$

If α is the result of a binary operator $\oplus$, function *Auto* is applied to the operands and the results are joined. If the operator is $\wedge$ or $\vee$, it is directly applied to the predicates generated by the two branches. For the operators **R** and **U**, the functions *Release* and *Until* are applied.

$\mathit{Auto}(\beta \oplus \gamma, m) =$
  $(H_1, D_1, p) := \mathit{Auto}(\beta, m)$;
  $(H_2, D_2, q) := \mathit{Auto}(\gamma, p)$;
  $H_3 := H_1 \sqcup H_2$;
  if $\oplus = \wedge$ **then return** $(H_3, D_1 \wedge D_2, q)$
  elsif $\oplus = \vee$ **then return** $(H_3, D_1 \vee D_2, q)$
  elsif $\oplus =$ **R then** $(H_4, P) := \mathit{Release}(H_3, D_1, D_2, q)$;
    return $(H_4, P, q + 1)$
  else $\{\oplus =$ **U**$\}$ $(H_4, P) := \mathit{Until}(H_3, D_1, D_2, q)$;
    return $(H_4, P, q + 1)$.



A picture has been drawn to suggest the structure of the construction, and the reason for the nomenclature.

Remark. As $\Box$ and $\Diamond$ are represented via $\Box\varphi = \bot \mathbf{R} \varphi$ and $\Diamond\varphi = \top \mathbf{U} \varphi$, one may introduce the convention $P_0 = \emptyset$, $P_1 = X$. The symbols (0) and (1) then get the reserved meanings $\bot$ and $\top$, and the base case of *Auto* is redefined as $\mathit{Auto}((i), m) = (\mathit{Triv}, Q'_i, m)$ with $Q'_i = (i = 0 ? \emptyset : i = 1 ? W : Q_i)$. ♣

5.4. Houses

The proof of correctness of the construction has two main aspects: the syntactic question of non-interference of the constructions for subformulas, and the semantic task of implementing a property. First non-interference. The frames of the skeletons are used to ensure that the constructions for the subformulas do not interfere.

For any finite set $F \subseteq \mathbb{N}$, the subset $j.F$ of W is defined by

$$j.F = \{(y, z) \in W \mid y \subseteq F\} .$$

Note that in $j.\emptyset$ the auxiliary variables are all false, but the propositional variables are still available.

The natural projection function $\pi_F : W \to W$ is defined by $\pi_F(y, z) = (y \cap F, z)$. Note that $\pi_F \circ \pi_G = \pi_{F \cap G}$ for subsets F and G of $\mathbb{N}$.

A skeleton $H = (N, \mathcal{V}, F)$ is interpreted by its associated specification $\sigma.H = (W, N, \varphi_1 \wedge \varphi_2)$ where $\varphi_1 = \Box[\![j.F]\!]$ and $\varphi_2 = \bigcap_{V \in \mathcal{V}} \Box\Diamond[\![V]\!]$. In words, runs are required to remain in the subset $j.F$ and to visit every $V \in \mathcal{V}$ infinitely often. It is convenient to note that $\varphi_1(ys) \equiv (ys = \pi_F \circ ys)$. For simplicity of notation, we define $\mathbf{run}.H = \mathbf{run}.(\sigma.H)$; a run of H is defined to be a run of $\sigma.H$.

Remark. As the finite subset $j.F$ is an invariant of the specification, it may be regarded as a proxy for the state space. ♣

Lemma 10 Let H be a skeleton and φ a property with $\mathbf{run}.H \subseteq \varphi$. Then $\mathbf{run}.H \subseteq \Box\varphi$.

Proof Let $xs \in \mathbf{run}.H$ and $k \in \mathbb{N}$. One needs to prove $(xs \mid k) \in \varphi$. This holds, because $(xs \mid k) \in \mathbf{run}.H$ in the absence of initialization. □

A predicate P on W is defined to *live on* F, notation $P \lhd F$, iff $P = P \circ \pi_F$. In the same way, a binary relation R is said to *live on* F, notation $R \lhd F$, iff $R = \pi_F^\bullet R$, and a property φ is said to *live on* F, notation $\varphi \lhd F$, iff $\varphi = \pi_F^\bullet \varphi$ (see Sect. 4.1).

A skeleton $(N, \mathcal{V}, F)$ is called a *house* iff all its components live on F, i.e., iff it satisfies

$$N \lhd F \ \wedge \ (\forall\ V \in \mathcal{V} : V \lhd F) . \tag{5}$$

An automaton (H, D) is called *proper* iff H is a house and $D \lhd F$ for the frame F of H.

The trivial skeleton *Triv* is a house. It is straightforward to verify the following result.

Lemma 11 Let $H = (N, \mathcal{V}, F)$ and $H' = (N', \mathcal{V}', G)$ be houses. Then the join $H \sqcup H'$ is also a house. An infinite sequence ws in W is a run of $H \sqcup H'$ if and only if $\pi_F \circ ws$ is a run of H and $\pi_G \circ ws$ is a run of H' and $ws \in \Box[\![j.(F \cup G)]\!]$.

Especially important is the join of houses with disjoint frames. Indeed, the next result shows that it is a kind of Cartesian product.

Lemma 12 Let $H = (N, \mathcal{V}, F)$ and $H' = (N', \mathcal{V}', G)$ be houses such that the frames F and G are disjoint. Let $us \in \mathbf{run}.H$ and $vs \in \mathbf{run}.H'$ have $p_2 \circ us = p_2 \circ vs$. Then there is a unique $ws \in \mathbf{run}.(H \sqcup H')$ with $\pi_F \circ ws = us$ and $\pi_G \circ ws = vs$.

Proof One first verifies that, if $u \in j.F$ and $v \in j.G$ have $p_2.u = p_2.v$, there is a unique element $w \in j(F \cup G)$ with $\pi_F.w = u$ and $\pi_G.w = v$. It follows that there is a unique sequence ws of states in $j(F \cup G)$ with $\pi_F \circ ws = us$ and $\pi_G \circ ws = vs$. This is a run of $H \sqcup H'$ because of Lemma 11. □

5.5. Star-houses

The semantics of the construction is analyzed by considering how properties on space W are treated.

A *star-house* is a pair $H * \varphi$ where H is a house and φ is a property on W that lives on the frame of H. If $H = (N, \mathcal{V}, F)$, the semantics of $H * \varphi$ is given by the specification $\sigma(H * \varphi) = (W, N, \varphi \wedge \varphi_1 \wedge \varphi_2)$ where $\varphi_1 = \Box[\![j.F]\!]$ and $\varphi_2 = \bigcap_{V \in \mathcal{V}} \Box\Diamond[\![V]\!]$, as before. It follows that

$$\mathbf{run}(H * \varphi) \quad = \quad \varphi \cap \mathbf{run}.H .$$

The extension relation for skeletons is extended to star-houses in the following way.

Star-house $H * \varphi$ is defined to be *extended by* $H' * \psi$, notation $H * \varphi \sqsubseteq H' * \psi$ iff

(a) $H \sqsubseteq H'$,
(b) $\forall\ ys \in \mathbf{run}(H' * \psi) : \pi_F \circ ys \in \mathbf{run}(H * \varphi)$,
(c) $\forall\ xs \in \mathbf{run}.H : \exists\ ys \in \mathbf{run}.H' : \pi_F \circ ys = xs\ \wedge\ ys \in \Box(\varphi \Rightarrow \psi)$,

where F is the frame of house H, and where, as usual, $\varphi \Rightarrow \psi$ is defined as $\neg\varphi \vee \psi$. In condition (c), the operator $\Box$ is needed, because it is needed in Formula (6) below.

The relevance of relation $\sqsubseteq$ is shown in the following result.

Lemma 13 (a) Assume that $H * \varphi \sqsubseteq H' * \psi$ and that F is the frame of H. Then function π_F is a birefinement $\sigma(H' * \psi) \to \sigma(H * \varphi)$.
(b) Relation $\sqsubseteq$ between star-houses is reflexive and transitive.

Proof (a) The proof is straightforward, but rather tedious.

(b) Reflexivity is easy. Transitivity is obvious for condition (a), and easy for (b). Condition (c) is proved as follows. Assume

$$(H_1 * \varphi) \sqsubseteq (H_2 * \psi) \sqsubseteq (H * \chi) .$$

Let F and G be the frames of H_1 and H_2, respectively. Let xs be a run of H_1. By the first extension, H_2 has a run ys such that $\pi_F \circ ys = \pi_F \circ xs$ and $ys \in \Box(\neg\varphi \vee \psi)$. The second extension implies that H has a run zs such that $\pi_G \circ zs = \pi_G \circ ys$ and $zs \in \Box(\neg\psi \vee \chi)$. As $F \subseteq G$, it holds that $\pi_F \circ zs = \pi_F \circ ys = \pi_F \circ xs$. It therefore suffices to prove that $zs \in \Box(\neg\varphi \vee \chi)$.

Let $k \in \mathbb{N}$. Then $(ys \mid k) \in \neg\varphi \vee \psi$ and $(zs \mid k) \in \neg\psi \vee \chi$. As $F \subseteq G$, both φ and ψ live on G, i.e., $\varphi = \pi_G^\bullet \varphi$ and $\psi = \pi_G^\bullet \psi$. It follows that $\pi_G \circ (ys \mid k) \in \neg\varphi \vee \psi$. As $\pi_G \circ zs = \pi_G \circ ys$, it follows that $\pi_G \circ (zs \mid k) \in \neg\varphi \vee \psi$, and hence that $(zs \mid k) \in \neg\varphi \vee \psi$. As $(zs \mid k) \in \neg\psi \vee \chi$, this implies $(zs \mid k) \in \neg\varphi \vee \chi$, as required. □

The join of star-houses is used for the treatment of the binary temporal operators $\wedge$, $\vee$, **R**, **U**. A binary temporal operator $\oplus$ is defined to be *monotonic* iff, for all properties $\varphi_1, \varphi_2, \psi_1, \psi_2$ on W, one has the inclusion of properties

(6) $$(\Box(\varphi_1 \Rightarrow \psi_1)\ \wedge\ \Box(\varphi_2 \Rightarrow \psi_2))\ \subseteq\ \Box(\varphi_1 \oplus \varphi_2 \Rightarrow \psi_1 \oplus \psi_2) .$$

A binary temporal operator $\oplus$ is defined to *preserve frames* iff, for every finite set F and all properties φ and ψ, it holds that

$$\varphi \lhd F\ \wedge\ \psi \lhd F\ \Rightarrow\ (\varphi \oplus \psi) \lhd F .$$

The binary temporal operators $\wedge$, $\vee$, **U**, **R** are all monotonic and preserve frames. This is used via the following result.

Lemma 14 Let H_1 and H_2 be houses with disjoint frames. Let $\varphi_1, \varphi_2, \psi_1, \psi_2$ be properties such that $\mathit{Triv} * \varphi_1$, $\mathit{Triv} * \varphi_2$, $H_1 * \psi_1$, $H_2 * \psi_2$ are star-houses. Assume $\mathit{Triv} * \varphi_1 \sqsubseteq H_1 * \psi_1$ and $\mathit{Triv} * \varphi_2 \sqsubseteq H_2 * \psi_2$. Let $\oplus$ be a monotonic binary operator that preserves frames. Then $\mathit{Triv} * (\varphi_1 \oplus \varphi_2) \sqsubseteq (H_1 \sqcup H_2) * (\psi_1 \oplus \psi_2)$ and these objects are star-houses.

The proof of this result uses primarily Lemma 12. At a critical point, it needs Lemma 10. It is mainly a cumbersome verification.

5.6. Proofs for the constructions for release and until

This section contains and proves the results for the functions *Release* and *Until*, defined in Sect. 5.3. In either case, it concerns the automaton for the property obtained when the operator **R** or **U** is applied to atomic properties.

Lemma 15 Let $H = (N, \mathcal{V}, F)$ be a house. Let P and Q be predicates on W that live on F. Consider the property $\varphi = [\![P]\!] \mathbf{R} [\![Q]\!]$. Let $m \notin F$ be chosen. Assume that $(H', D) = \mathit{Release}(H, P, Q, m)$. Then (H', D) is a proper automaton and $H * \varphi$ and $H' * [\![D]\!]$ are star-houses with $H * \varphi \sqsubseteq H' * [\![D]\!]$.

Proof Recall that

$$xs \in \varphi\quad \equiv\quad \forall\ k : Q(xs_k) \vee (\exists\ i : i < k \wedge P(xs_i)) .$$

As the predicates P and Q live on F, the property φ also lives on F. Therefore (H, φ) is a star-house. It is easy to verify that H' is a house. Predicate $D = (B_m \wedge Q)$ lives on $F \cup \{m\}$, which is the frame of H'. Therefore $H' * [\![D]\!]$ is a star-house. It is clear that $H \sqsubseteq H'$, i.e., condition (a).

For the proof of condition (b), consider a run ys of $H' * [\![D]\!]$. Then ys is a run of H' and hence of H. It satisfies $B_m(ys_0) \wedge Q(ys_0)$ because $ys \in [\![D]\!]$. As it is an execution of H', it satisfies, for all k,

$$B_m(ys_k) \wedge Q(ys_k) \wedge \neg P(ys_k) \;\Rightarrow\; B_m(ys_{k+1}) \wedge Q(ys_{k+1}) \,.$$

By induction, it follows that

$$\forall\ k : B_m(ys_k) \wedge Q(ys_k) \;\vee\; (\exists\ i : i < k \wedge P(ys_i)) \,.$$

This proves that $ys \in \varphi$.

For part (c), consider a run xs of H. A run ys of H' is constructed from xs by modifying the bits of the auxiliary variable B_m. Let ys be the infinite sequence of states within $j.(F \cup \{m\})$ with $\pi_F \circ ys = xs$ and $B_m(ys_n) = \varphi(xs \mid n)$ for all $n \in \mathbb{N}$. It is straightforward to prove that ys is an execution of H'. As xs is a run of H, it follows that ys is a run of H'. By construction it satisfies $ys \in \Box(\neg\varphi \vee [\![D]\!])$. This proves that $H * \varphi \sqsubseteq H' * [\![D]\!]$. □

Lemma 16 Let $H = (N, \mathcal{V}, F)$ be a house. Let P and Q be predicates on W that live on F. Consider the property $\varphi = [\![P]\!] \mathbf{U} [\![Q]\!]$. Let $m \notin F$ be chosen. Assume that $(H', D) = \mathit{Until}(H, P, Q, m)$. Then (H', D) is a proper automaton and $H * \varphi$ and $H' * [\![D]\!]$ are star-houses with $H * \varphi \sqsubseteq H' * [\![D]\!]$.

Proof Recall that

$$xs \in \varphi \quad \equiv \quad \exists\ k : Q(xs_k) \;\wedge\; (\forall\ i : i < k \;\Rightarrow\; P(xs_i)) \,.$$

Again, it is clear that φ lives on F, that $H * \varphi$ is a star-house, that H' is a house with $H \sqsubseteq H'$, that D is a predicate that lives on $F \cup \{m\}$, and that $H' * [\![D]\!]$ is a star-house.

Let ys be a run of $H' * [\![D]\!]$. Then ys is a run of H', and hence of H. The run ys starts in $D = (B_m \wedge P) \vee Q$. By induction over k, one can prove that

$$\forall\ n : P(ys_n) \;\wedge\; B_m(ys_n) \;\vee\; (\exists\ i : i \leq n \;\wedge\; Q(ys_i)) \,.$$

As $ys \in \Box\Diamond[\![\neg B_m]\!]$ (see Sect. 2.6), there is an index n with $\neg B_m(ys_n)$. It follows that $ys_k \in Q$ for some index k. Taking the smallest index k, one gets $ys \in \varphi$.

Conversely, let xs be a run of H. The corresponding run ys of H' is constructed with $\pi_F \circ ys = xs$ with $B_m(ys_n) = \neg Q(xs_n) \wedge \varphi(xs \mid n)$ for all $n \in \mathbb{N}$. One then verifies that ys is a run of H' that satisfies $ys \in \Box(\neg\varphi \vee [\![D]\!])$. This concludes the proof that $H * \varphi \sqsubseteq H' * [\![D]\!]$. □

5.7. The automaton proved

Let the *weight* $w.\alpha$ of an LTL formula α be defined as the number of occurrences of **R** and **U** in the syntax tree of α. Write $I(m, r) = \{i \in \mathbb{N} \mid m \leq i < r\}$.

Theorem 17 Let α be an LTL formula. Assume $(H, D, r) = \mathit{Auto}(\alpha, m)$. Then (H, D) is a proper automaton with $\mathit{Triv} * [\![\alpha]\!] \sqsubseteq H * [\![D]\!]$. The frame of H is $I(m, r)$ and $r = m + w.\alpha$.

Proof The proof is by induction over the structure of α. If α is a leaf, the assertion is trivial. Therefore, assume that $\alpha = \beta \oplus \gamma$ for one of the binary temporal operators $\oplus$. Use the variables of the algorithm of Sect. 5.3. By induction it holds that $m \leq p \leq q$, that H_1 and H_2 have the disjoint frames $I(m, p)$ and $I(p, q)$, respectively, and that $\mathit{Triv} * [\![\beta]\!] \sqsubseteq H_1 * [\![D_1]\!]$ and $\mathit{Triv} * [\![\gamma]\!] \sqsubseteq H_2 * [\![D_2]\!]$. Lemma 14 implies that $\mathit{Triv} * [\![\alpha]\!] \sqsubseteq H_3 * ([\![D_1]\!] \oplus [\![D_2]\!])$, and H_3 has frame $I(m, q)$. If $\oplus$ is $\wedge$ or $\vee$, one can conclude with the observations that $[\![D_1]\!] \oplus [\![D_2]\!] = [\![D_1 \oplus D_2]\!]$ and $w.\alpha = w.\beta + w.\gamma$.

Otherwise $\oplus$ is one of the operators **R** or **U** and $w.\alpha = 1 + w.\beta + w.\gamma$. Then one applies Lemma 15 or 16 to obtain an automaton (H_4, P) with frame $I(m, q + 1)$ and the extension relation $H_3 * ([\![D_1]\!] \oplus [\![D_2]\!]) \sqsubseteq H_4 * [\![P]\!]$. One concludes with Lemma 13 (transitivity of $\sqsubseteq$). □

5.8. Implementing a U*-specification

The first thing to be done is to translate the automaton constructed in the Theorem into a U-specification. Recall that A_0 is the trivial U-specification with state space $Z = \mathbb{P}(\mathbb{N})$, which was introduced in the beginning of this section.

Corollary 18 Let α be an LTL formula with weight $w.\alpha = r$. Then there is a U-specification C with state space $X = \mathbb{B}^r \times Z$, and a predicate D on X, such that $p_2 : C * [\![D]\!] \to A_0 * [\![\alpha]\!]$ is an implementation, where p_2 is the projection on the second component.

Proof Let $(H, D', r) = \mathit{Auto}(\alpha, 0)$. Then the frame of H is $I(0, r)$. By Lemma 13(a), the function $\pi_\emptyset$ is a birefinement $H * [\![D']\!] \to \mathit{Triv} * [\![\alpha]\!]$. The automaton (H, D') is translated in a U-specification C on the state space X by means of the injection $f : X \to W$ given by $f.(x, z) = (x', z)$ where $x'(i) = (i < r \wedge x(i))$, and the projection $g : Z \to X$ given by $g(y, z) = (y', z)$ where $y' = (y \mid I(0, r))$. In particular, one takes $D = D' \circ f$. Then f induces a birefinement from $C * [\![D]\!]$ to $H * [\![D']\!]$. On the other hand, one constructs in a natural way a birefinement from $\mathit{Triv} * [\![\alpha]\!]$ to $A_0 * [\![\alpha]\!]$. The composition of the three birefinements is the birefinement p_2. □

Part 3 of the construction, the lifting of an implementation $g : C * [\![D_0]\!] \to A_0 * [\![\alpha]\!]$ to an implementation $E * [\![D]\!] \to A * \varphi$ is done by superposition. Formally, this is a fiber product construction.

In general, the definition is as follows. If X_0, X_1, and X_2 are sets with functions $f_i : X_i \to X_0$, the *fiber product of X_1 and X_2 over f_1 and f_2* is defined as the set $X = \{(x_1, x_2) \in X_1 \times X_2 \mid f_1.x_1 = f_2.x_2\}$.

Let $C_i = (X_i, N_i, \mathcal{W}_i)$, with $i = 1, 2$, be U-specifications on X_1 and X_2, respectively. The *fiber product* of C_1 and C_2 over f_1 and f_2 is defined as the U-specification $E = (X, N, \mathcal{W})$ on the fiber product X given by

$$N = \{(y, z) \in X^2 \mid (y_1, z_1) \in N_1 \wedge (y_2, z_2) \in N_2\} ,$$
$$\mathcal{W} = \{N \wedge p_1^\bullet R \mid R \in \mathcal{W}_1\} \cup \{N \wedge p_2^\bullet R \mid R \in \mathcal{W}_2\} ,$$

where $p_i : X \to X_i$ for $i = 1, 2$ are the canonical projection functions. Note that $f_1 \circ p_1 = f_2 \circ p_2$ holds by construction. One can say that the conjoined state space X is the consistent part of the Cartesian product of the component spaces, that the steps are done in parallel in both components, and that the fairness of the components is retained. It is easy to prove:

Lemma 19 (a) The functions $p_i : E \to C_i$ ($i = 1$ or 2) are refinement functions.
(b) If xs and ys are runs of C_1 and C_2 with $f_1 \circ xs = f_2 \circ ys$, there is a unique run zs of E with $p_1 \circ zs = xs$ and $p_2 \circ zs = ys$.

Now, part 3 of the construction is the special case, captured in the following diagram and proposition.

$$\begin{array}{ccc} E * [\![D]\!] & \xrightarrow[p_2]{} & C * [\![D_0]\!] \\ \downarrow{\scriptstyle p_1} & & \downarrow{\scriptstyle g} \\ A * \varphi & \xrightarrow{f} & A_0 * [\![\alpha]\!] \end{array}$$

Proposition 20 Let $A * \varphi$ be a U*-specification with state space X_1 and LTL property φ. Let α be an LTL-formula and $f : X_1 \to Z$ a function such that $\varphi = f^\bullet [\![\alpha]\!]$. Let $g : C * [\![D_o]\!] \to A_0 * [\![\alpha]\!]$ be an implementation. Let E be the fiber product of A and C over f and g, with state space X. Put $D = p_2^{-1} D_0 \subseteq X$. Then $p_1 : E * [\![D]\!] \to A * \varphi$ is an implementation.

Proof The proof obligation is that $p_1 : E * [\![D]\!] \to A * \varphi$ is a birefinement function.

Let ys be a run of $E * [\![D]\!]$. By Lemma 19(a), the sequences $p_1 \circ ys$ and $p_2 \circ ys$ are runs of A and C respectively. As ys starts in $D = p_2^{-1} D_0$, the sequence $p_2 \circ ys$ is a run of $C * [\![D_0]\!]$. As $g : C * [\![D_0]\!] \to A_0 * [\![\alpha]\!]$ is an implementation, it holds that $g \circ p_2 \circ ys \in [\![\alpha]\!]$. This implies $p_1 \circ ys \in \varphi$ because of $f \circ p_1 = g \circ p_2$ and the equality for φ. This proves that $p_1 : E * [\![D]\!] \to A * \varphi$ is a refinement function.

Let xs be a run of $A * \varphi$. As A_0 is the trivial U-specification, $f \circ xs$ is a run of A_0. On the other hand, $f \circ xs \in [\![\alpha]\!]$ because $xs \in \varphi$. Therefore $f \circ xs$ is a run of $A_0 * [\![\alpha]\!]$. As g is an implementation of $A_0 * [\![\alpha]\!]$, the U*-specifcation $C * [\![D_0]\!]$ has a run ys with $g \circ ys = f \circ xs$. By Lemma 19(b), there is a run zs of E with $p_1 \circ zs = xs$ and $p_2 \circ zs = ys$. It follows that zs starts in D. Therefore zs is a run of $E * [\![D]\!]$ that projects to xs. This proves that $p_1 : E * [\![D]\!] \to A * \varphi$ is a corefinement function. □

Corollary 18 and Proposition 20 together imply the final result:

Theorem 21 Let A be a U-specification on a state space X. Let φ be an LTL property on X. Then $A * \varphi$ has an implementation $E * [\![D]\!] \to A * \varphi$.

Remark. In the construction of Corollary 18, the state space X_2 of C is a Cartesian product $X_2 = Y \times Z$ and $g : X_2 \to Z$ is the projection onto the second factor. Therefore, the state space of the fiber product

$$X = \{(x, (y, z)) \in X_1 \times (Y \times Z) \mid f.x = z\}$$

is in a natural way isomorphic to the Cartesian product $X \times Y$, viz. via the projection $\lambda\, x, (y, z) : (x, y)$. In this way, the component Z is eliminated. It follows that one can give the U-specifcation E of the theorem the state space $X \times \mathbb{B}^r$ where r is the number of operators **R** and **U** that occur in φ. This has been done in the example in Sect. 4.3. ♣

6. Conclusions

It is striking how much smoother the UNITY theory becomes by postponing or omitting the initialization. Of course, in almost all applications, one begins with the determination of the inductive invariants, which requires the initialization as a starting point.

UNITY is a powerful and flexible formalism, especially if one allows the fairness relations to be nontotal. In the completeness proof for L-specifications, the nontotal fairness relations only play an auxiliary role. They are, however, essential for the construction of implementations of LTL formulas.

The algorithm to construct a Büchi automaton for an LTL formula in Sect. 5.3 is shorter and clearer than Figure 1 of Gerth et al. [GPVW95], but it can make the state space bigger than necessary. The state space has a clear relationship with the LTL formula: there is precisely one boolean variable for each temporal operator. The termination of our algorithm is obvious, whereas Schimpf et al. [SMS09] need a page to discuss the proof of termination of the algorithm of Gerth et al.

The construction of an implementation of an LTL property in Sect. 5 uses auxiliary variables. This may remind the reader of the Completeness Theorem of Abadi and Lamport [AL91], which asserts that, under certain conditions, every simulation between specifications can be factored over an extension with history variables and prophecy variables followed by a refinement mapping. The three kinds of variables and their roles in the constructions, however, are totally different.

Points for future research. In view of the example in Sect. 2.6, the difference between U-specifications and B-specifications deserves further investigation. For Sect. 4, a useful application of Theorem 8 would be very illuminating. When such applications are found, it will be useful to investigate LTL formulas that have implementations simpler than the ones constructed in Theorem 21. It should be possible to extend LTL and the results of Sects. 4 and 5 with the infinitary operators for conjunction and disjunction $\bigwedge$ and $\bigvee$. It seems that, even after this extension, the set of formulas that can be subjected to Theorem 8 is not yet exhausted, but natural candidates are lacking.

Chandy, Dijkstra, and Sanders [CS95, DS97] extend UNITY logic with predicate transformers for 'to-stable' and 'to-always'. While 'to-always' has a natural interpretation in LTL, 'to-stable' seems to require a more expressive form of temporal logic, e.g., as CTL. Once the operational semantics has been clarified, it should not be too difficult to prove soundness of these predicate transformers. Their names are only justified, however, if their completeness can be proved.

Would it be possible to implement parts of branching temporal logic (e.g. CTL) in UNITY? This is related to the search for methods or rules to prove that a temporal property is invalid. Of course, a model checker may produce a counterexample, but should we be content with this?

Acknowledgements

I am indepted and very grateful to Ernie Cohen who taught me the main results of this paper many years ago. We had planned to make this a joint paper, but circumstances intervened. He now even abstains from being a coauthor.

References

[AL91] Abadi M, Lamport L (1991) The existence of refinement mappings. Theor Comput Sci 82:253–284
[BK08] Baier C, Katoen JP (2008) Principles of model checking. MIT Press, Cambridge
[BKS88] Back R-J, Kurki-Suonio R (1988) Distributed cooperation with action systems. ACM Trans Program Lang Syst 10:513–554
[Bue60] Buechi JR (1960) On a decision method in restricted second order arithmetic. In: Proceedings of international congress on logic, methodology and philosophy of science, pp 1–11. Stanford University Press
[CK97] Collette P, Knapp E A (1997) foundation for modular reasoning about safety and progress properties of state-based concurrent programs. Theor Comput Sci 183:253–279
[CM88] Chandy KM, Misra J (1988) Parallel program design, a foundation. Addison–Wesley, Reading, MA
[CS95] Chandy KM, Sanders BA (1995) Predicate transformers for reasoning about concurrent computation. Sci Comput Program 24:129–148
[Dij76] Dijkstra EW (1976) A discipline of programming. Prentice Hall, Englewood cliffs
[Dij95] Dijkstra RM (1995) DUALITY: a simple formalism for the analysis of UNITY. Formal Aspects Comput 7:353–388
[Dij00] Dijkstra RM (2000) Computation calculus bridging a formalization gap. Sci Comput Program 37:3–36
[DS97] Dijkstra RM, Sanders BA (1997) A predicate transformer for the progress property 'to-always'. Formal Aspects of Comput 9:270–282
[GP89] Gerth R, Pnueli A (1989) Rooting UNITY. In: Proceedings of the 5th international workshop on software specification and design, pp 11–19. ACM
[GPVW95] Gerth R, Peled D, Vardi MY, Wolper P (1995) On-the-fly automatic verification of linear temporal logic. In: Proceedings 15th workshop on protocol specification, testing, and verification, pp 3–18, Chapman & Hall, Warsaw, Poland.
[GZ96] Gumm HP, Zhukov D (1996) On strong fairness in UNITY. Technical Report, Philipps Universität Marburg
[Hes92] Hesselink WH (1992) Programs, recursion and unbounded choice, predicate transformation semantics and transformation rules. Cambridge University Press, Cambridge
[Hes13] Hesselink WH (2013) Complete assertional proof rules for progress under weak and strong fairness. Sci Comput Program 78:1521–1537. doi:10.1016/j.scico.2012.10.013
[Hes20] Hesselink WH (2020) PVS proof script for: UNITY and Büchi automata. http://wimhesselink.nl/mechver/unity-and-beyond
[Hol04] Holzmann GJ (2004) The SPIN model checker, primer and reference manual. Addison-Wesley, Reading
[JKR89] Jutla CS, Knapp E, Rao JR (1989) A predicate transformer approach to semantics of parallel programs. In: Proceedings of 8th annual ACM symposium on principles of distributed computing. ACM Press, pp 249–263
[Kna92] Knapp E (1992) Refinement as a basis for concurrent program design. PhD thesis, The University of Texas at Austin
[Kna94] Knapp E (1994) Soundness and completeness of UNITY logic. In: Thiagarajan PS, (ed) Foundations of software technology and theoretical computer science, vol 880 of *LNCS*. Spinger, pp 378–389
[Lam83] Lamport L (1983) What good is temporal logic? Inf Process 83:657–668
[Lam94] Lamport L (1994) The temporal logic of actions. ACM Trans Program Lang Syst 16:872–923
[Mis01] Misra J (2001) A discipline of multiprogramming: programming theory for distributed applications. Spinger, New York
[Mor88] Morgan C (1987/88) Data refinement by miracles. Inf Process Lett, pp 243–246
[MP83] Manna Z, Pnueli A (1983) How to cook a temporal proof system for your pet language. In: Proceedings 10th annual symposium on principles of programming languages (POPL). ACM, pp 141–154
[SMS09] Schimpf A, Merz S, Smaus J-G (2009) Construction of Büchi automata for LTL model checking verified in Isabel/HOL. In: International conference on theorem proving in higher order logics, TPHOLs 2009, vol 5674 of LNCS, Springer, New York, pp 424–439.
[TB95] Tsay Y-K, Bagrodia RL (1995) Deducing fairness properties in UNITY logic: a new completeness result. ACM Trans Program Lang Syst 17:16–27

Received 7 May 2020
Accepted in revised form 19 December 2020 by Eerke Albert Boiten
Published online 10 February 2021