Contents

Chapter 5

Temporal Safety Requirements

5.1 Logical Requirements of Reactive Modules

The invariant-verification problem allows us to check if all reachable states of a reactive module satisfy an observation predicate. Not all module requirements can be formulated as invariants. For instance, for the mutual-exclusion protocols of Chapter 2, we may wish to check the first-request-first-in requirement, that the first process to request admission to the critical section is the first process allowed to enter the critical section. As was discussed in Chapter 2, the first-request-first-in requirement can be checked by composing the mutual-exclusion protocol with a monitor and verifying an invariant of the compound module. The introduction of monitors has the disadvantage of increasing the state space of a module. Here we discuss the alternative of enriching the specification language so that module requirements such as first-request-first-in can be formulated without the use of monitors. For this purpose, we present a class of formal languages called state logics. While invariants refer to static observation snapshots of reactive modules, the formulas of state logics refer also to dynamic observation sequences. The observation sequences of a module are captured by an observation structure.

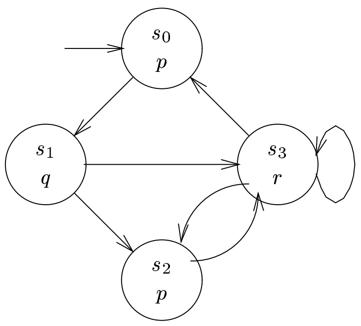

Figure 5.1: Observation structure

5.1.1 Observation Structures

An observation structure is a transition graph whose states are labeled with observations. It is required that there are only finitely many initial states with a given observation, and every state has only finitely many successors with a given observation.

Observation structure An observation structure K consists of (1) a transition graph $(\Sigma, \sigma^I, \rightarrow)$, (2) [the observation alphabet] a set A of observations, and (3) [the observation function] a function $\langle\langle\cdot\rangle\rangle : \Sigma \rightarrow A$ that maps each state s to an observation $\langle\langle s\rangle\rangle$ such that for every observation $a \in A$, (i) the set $\{s \in \sigma^I \mid \langle\langle s\rangle\rangle = a\}$ is finite, and (ii) for every state $s \in \Sigma$, the set $\{t \in post_G(s) \mid \langle\langle t\rangle\rangle = a\}$ is finite.

Example 5.1 [] Observation structure Figure 5.1 shows an observation structure with the state space $\{s_0, s_1, s_2, s_3\}$ and the observation alphabet $\{p, q, r\}$. ■

We freely attribute properties and derivatives of transition graphs to observation structures. For example, an observation structure inherits the trajectories of the underlying transition graph, and the reachable subgraph of the underlying transition graph gives the reachable substructure of an observation structure.

The observation structure of a module

Every reactive module P defines the transition graph G_P. An observation of P is a valuation to the set $\mathsf{obs}X_P = \mathsf{intf}X_P \cup \mathsf{extl}X_P$ of observable variables of P. The observation alphabet is, then, the set of all observations. The observation function maps a module state s to the projection $s[\mathsf{obs}X_P]$ to the observable variables. There are only finitely many ways to initialize the controlled state of a module: for every valuation t of the external variables $\mathsf{extl}X_P$, there are only finitely many initial states s such that $s[\mathsf{extl}X_P] = t$. It follows that $\sigma_P^I[\mathsf{obs}X_P]$ is finite. Similarly, there are only finitely many ways to update the controlled state of a module: for a state s and a valuation t of the external variables $\mathsf{extl}X_P$, there are only finitely many states u such that $s \rightarrow_P u$ and $u[\mathsf{extl}X_P] = t$. It follows that $post_P(s)[\mathsf{obs}X_P]$ is finite for every state s of the module.

Observation structure of a module
The reactive module P defines the observation structure $K_P = (G_P, \Sigma_{\mathsf{obs}X_P}, \cdot[\mathsf{obs}X_P])$.

Example 5.2 [] Observation structure of Pete Recall Peterson's mutual exclusion protocol from Chapter 2. The observable variables for the module *Pete* are the location variables pc_1 and pc_2. The observation alphabet of the observation structure K_{Pete} is

$$\Sigma_{\{pc_1, pc_2\}} = \{inC, reqC, outC\} \times \{inC, reqC, outC\}.$$

The reachable substructure of K_{Pete} has the transition graph of Figure 3.4, and each state is labeled with the value of pc_1 and pc_2. Note that the transition graph of the module $P_1 \,\|\, P_2$ is identical to the transition graph of the module $Pete = \textbf{hide } x_1, x_2 \textbf{ in } P_1 \,\|\, P_2$, but the observation structures $K_{P_1 \| P_2}$ and K_{Pete} have different observation alphabets, and thus, are nonisomorphic. ■

5.1.2 State Logics

The formulas of state logics are called state formulas, because they are interpreted over the states of observation structures; that is, a state formula is true or false in a given state of a given observation structure. Before we study specific state logics, let us define concepts generic to all state logics. A state logic is defined by specifying the rules to build formulas of the logic, and the rules to interpret formulas at states of an observation structure. A state logic Φ consists of two components:

Syntax A set $\mathcal{P}_\Phi$ of formulas. Formulas are defined inductively from atomic formulas using boolean and temporal connectives. The formulas in $\mathcal{P}_\Phi$ are called Φ-formulas.

Semantics Given a Φ-formula ϕ, an observation structure K is said to be a ϕ-structure if each observation of K is a valuation for a superset of the variables appearing in the atomic formulas of ϕ. Every Φ-formula ϕ is interpreted over the states of the ϕ-structures. The satisfaction relation $s \models_K \phi$ specifies when the state s of the ϕ-structure K satisfies the Φ-formula ϕ.

The satisfaction relation specifies the truth of a formula at a state of an observation structure. An observation structure satisfies a formula if all its initial states do.

Satisfiability

Let ϕ be a state formula, and let K be a ϕ-structure with the state space Σ and the initial region σ^I. The characteristic ϕ-region of K is the region

$$[\![\phi]\!]_K \;=\; \{s \in \Sigma \mid s \models_K \phi\}$$

of states in Σ that satisfy the formula ϕ. The observation structure K satisfies the formula ϕ, written $K \models \phi$, if all initial states in σ^I satisfy ϕ; that is, $\sigma^I \subseteq [\![\phi]\!]_K$. The state formula ϕ is satisfiable if there is a ϕ-structure that satisfies ϕ. The state formula ϕ is valid if all ϕ-structures satisfy ϕ.

The satisfiability problem for a state logic is to decide whether a given formula is satisfiable, and the validity problem is to decide whether a given formula is valid. Let ϕ and ψ be two state formulas. If the observation structure K is both a ϕ-structure and a ψ-structure, we say that K is a (ϕ,ψ)-structure. The state formula ϕ implies the state formula ψ if for every (ϕ,ψ)-structure K, the region $[\![\phi]\!]_K$ is a subset of the region $[\![\psi]\!]_K$. The two state formulas ϕ and ψ are equivalent if for every (ϕ,ψ)-structure K, the regions $[\![\phi]\!]_K$ and $[\![\psi]\!]_K$ are equal.

Exercise 5.1 {} [] Weak equivalence of state formulas Let ϕ and ψ be two state formulas. The two state formulas ϕ and ψ are weakly equivalent if for every (ϕ,ψ)-structure K, the formula ϕ is K-valid iff the formula ψ is K-valid. Prove that ϕ and ψ are equivalent iff they are weakly equivalent. ■

The model-checking problem

The model-checking problem for Φ asks if ϕ is satisfied by a given ϕ-structure.

Model-checking problem for state logics

An instance (K,ϕ) of the model-checking problem for the state logic Φ consists of (1) a Φ-formula ϕ and (2) a ϕ-structure K. The answer to the model-checking problem (K,ϕ) is Yes if K satisfies ϕ, and otherwise No.

The verification problem for a state logic is to check whether the observation structure of a module satisfies a given formula. It can be reduced, at an exponential cost, to the model-checking problem, similar to the way in which the invariant-verification problem can be reduced to the reachability problem.

Verification problem for state logics

An instance (P, ϕ) of the verification problem for the state logic Φ consists of (1) a reactive module P with the set $\mathsf{obs}X_P$ of observable variables, and (2) a Φ-formula ϕ such that the set of variables appearing in the atomic formulas of ϕ is a subset of $\mathsf{obs}X_P$. The answer to the verification problem (P, ϕ) is the answer to the model-checking problem (K_P, ϕ).

5.2 Safe Temporal Logic

Temporal logics extend observation logic with connectives that refer to observation sequences of reactive modules.

5.2.1 Syntax and Semantics of Stl

Safe temporal logic has two temporal connectives: the unary connective possibly-next, written $\exists\bigcirc$, and the binary connective possibly-until, written $\exists\mathcal{U}$.

Safe temporal logic: Syntax

Safe temporal logic (Stl) is the state logic whose formulas are generated by the grammar

$$\phi \ ::= \ p \mid \phi \vee \phi \mid \neg\phi \mid \exists\bigcirc \phi \mid \phi\exists\mathcal{U}\phi,$$

for atomic formulas p.

The satisfaction of an atomic formula p at a state s depends on the observation of s. The meaning of the boolean connectives is the usual one. Consider two observation predicates p and q for a reactive module P. The state s of the observation structure K_P satisfies the formula $\exists\bigcirc p$ if some successor of s satisfies p. The state s satisfies the formula $q\exists\mathcal{U}p$ if there is a source-s trajectory whose states satisfy p or q, and whose sink satisfies p. In other words, the formula $\exists\bigcirc p$ asserts that it is possible to execute a single round of the module P so that the observation predicate p becomes true. The formula $q\exists\mathcal{U}p$ asserts that it is possible to execute finitely many rounds of P so that p becomes true, and throughout the execution the observation predicate $p \vee q$ is true.

Safe temporal logic: Semantics

The satisfaction relation for Stl is defined inductively by the following clauses:

$s \models_K p$	iff	$\langle\langle s \rangle\rangle \models p$;
$s \models_K \phi \vee \psi$	iff	$s \models_K \phi$ or $s \models_K \psi$;
$s \models_K \neg\phi$	iff	$s \not\models_K \phi$;
$s \models_K \exists\bigcirc\, \phi$	iff	there is a state $t \in post_K(s)$ such that $t \models_K \phi$;
$s \models_K \psi \exists\mathcal{U} \phi$	iff	there is a source-s trajectory $\overline{s}_{0..m}$ of K such that (1) $s_m \models_K \phi$ and (2) for all $0 \leq i \leq m$, $s_i \models_K \phi \vee \psi$.

where p is an atomic formula, ϕ and ψ are Stl-formulas, K is a (p, ϕ, ψ)-structure, and s is a state of K.

The propositional connectives $\wedge$ (conjunction), $\rightarrow$ (implication), and $\leftrightarrow$ (equivalence) can be defined using the connectives $\vee$ (disjunction) and $\neg$ (negation) of state logics, and we freely use the defined connectives as abbreviations.

Remark 5.1 [Until connective] Equivalently, $s \models_K \psi\exists\mathcal{U}\phi$ iff there is a source-s trajectory $\overline{s}_{0..m}$ of K such that (1) $s_m \models_K \phi$ and (2) for all $0 \leq i < m$, $s_i \models_K \psi$. In particular, if $s \models_K \phi$, then $s \models_K \psi\exists\mathcal{U}\phi$. ■

Example 5.3 [] Safe temporal logic Recall the observation structure shown in Figure 5.1. There, $s_0 \models \exists\bigcirc\, q$ and $s_0 \models (p \vee q)\exists\mathcal{U}r$. ■

When all the atomic formulas are propositional formulas, that is, boolean expressions over boolean variables, the temporal logic Stl is called propositional Stl.

Defined temporal connectives

Using the connectives $\exists\bigcirc$ and $\exists\mathcal{U}$, we can define additional temporal connectives in Stl:

Inevitably-next	$\forall\bigcirc\, \phi$	for	$\neg\, \exists\bigcirc\, \neg\phi$;
Possibly-eventually	$\exists\Diamond\phi$	for	$true\, \exists\mathcal{U}\phi$;
Inevitably-always	$\forall\Box\phi$	for	$\neg\, \exists\Diamond\neg\phi$;
Inevitably-waiting-for	$\phi\forall\mathcal{W}\psi$	for	$\neg((\neg\psi)\exists\mathcal{U}\neg(\phi \vee \psi))$.

Consider two observation predicates p and q for a reactive module P. The state s of the observation structure K_P satisfies the formula $\forall\bigcirc\, p$ if every successor of s satisfies p. The state s satisfies the formula $\exists\Diamond p$ if some state in the sink region of s satisfies p, and s satisfies $\forall\Box p$ if every state in the sink region of s satisfies p. In other words, the formula $\forall\bigcirc\, p$ asserts that after executing a

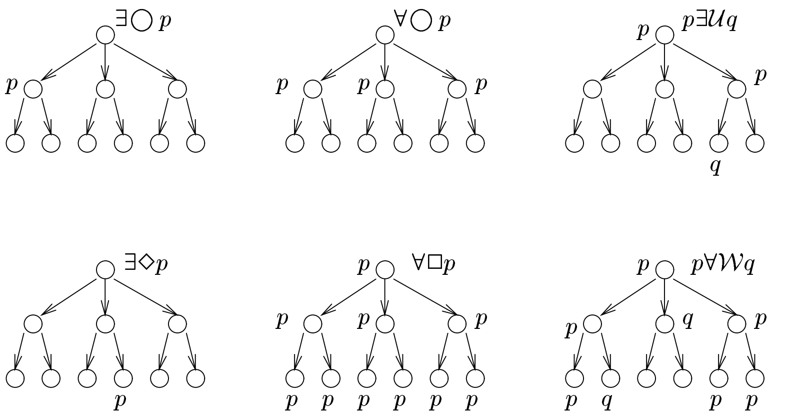

Figure 5.2: The temporal connectives of Stl

single round of the module P the observation predicate p becomes true. The formula $\exists\Diamond p$ asserts that it is possible to execute finitely many rounds of P so that p becomes true, and the formula $\forall\Box p$ asserts that p is an invariant of P. It follows that the invariant-verification problem is a special case of the verification problem for Stl: the invariant-verification problem (P, p) and the Stl-verification problem $(P, \forall\Box p)$ have the same answer.

Exercise 5.2 {} [Waiting-for connective] Let ϕ and ψ be two Stl formulas, let K be a (ϕ, ψ)-structure, and let s be a state of K. Prove that $s \models_K \phi\forall\mathcal{W}\psi$ iff for all source-s trajectories $\overline{s}_{0..m}$ of K either for all $0 \leq i \leq m$, $s_i \models_K \phi$; or there is a natural number j with $0 \leq j \leq m$ such that (1) $s_j \models_K \psi$ and (2) for all $0 \leq i < j$, $s_i \models_K \phi$. That is, in every source-s trajectory of K, a state that violates ϕ coincides with or is preceded by a state that satisfies ψ. ■

Figure 5.2 graphically illustrates the requirements that are imposed by the temporal connectives of Stl. When writing Stl formulas, we freely use the defined temporal connectives as abbreviations. We suppress parentheses, assuming that the binary connectives $\exists\mathcal{U}$ and $\forall\mathcal{W}$ associate to the right; that is, we write $\phi\exists\mathcal{U}\psi\exists\mathcal{U}\chi$ for $\phi\exists\mathcal{U}(\psi\exists\mathcal{U}\chi)$.

Exercise 5.3 {} [Nested until connectives] Let ϕ, ψ, and χ be three Stl formulas. Let K be a (ϕ, ψ, χ)-structure, and let s be a state of K. Prove that $s \models_K \phi\exists\mathcal{U}(\psi\exists\mathcal{U}\chi)$ iff there is a source-s trajectory $\overline{s}_{0..m}$ of K and a natural number i with $0 \leq i \leq m$ such that (1) $s_m \models_K \chi$, (2) for all $i \leq j < m$, $s_j \models_K \psi$, and (3) for all $0 \leq j < i$, $s_j \models_K \phi$.

Prove or disprove that $(\phi\exists\mathcal{U}\psi)\exists\mathcal{U}\chi$ implies $\phi\exists\mathcal{U}(\psi\exists\mathcal{U}\chi)$, and vice versa. What about $(\phi\forall\mathcal{W}\psi)\forall\mathcal{W}\chi$ vs. $\phi\forall\mathcal{W}(\psi\forall\mathcal{W}\chi)$? ■

Exercise 5.4 {} [Dual version of Stl] Suppose that the Stl$^{\forall}$ formulas are generated by the grammar

$$\phi \ ::= \ p \mid \phi \vee \phi \mid \neg\phi \mid \forall\bigcirc\, \phi \mid \phi \forall\mathcal{W}\phi.$$

Define the temporal connectives $\exists\bigcirc$ and $\exists\mathcal{U}$ in this logic; that is, for every Stl formula give an equivalent Stl$^{\forall}$ formula. ■

5.2.2 Specifying requirements using Stl

Example 5.4 [Mutual exclusion] Recall Peterson's mutual-exclusion protocol from Chapter 2. The mutual-exclusion requirement is specified by the Stl formula

$$\forall\Box\,\neg\,(pc_1 = inC \ \wedge \ pc_2 = inC), \qquad (\phi_{mutex})$$

which is equivalent to the Stl formula

$$\neg\,\exists\Diamond\,(pc_1 = inC \ \wedge \ pc_2 = inC).$$

The Stl formula ϕ_{fifo} specifies the first-request-first-in requirement that if process P_1 attempts to enter the critical section when process P_2 is in its noncritical section , then P_2 cannot overtake P_1 to enter the critical section:

$$\forall\Box((pc_1 = reqC \ \wedge \ pc_2 = outC) \ \rightarrow \ (pc_2 \neq inC)\forall\mathcal{W}(pc_1 = inC)).$$

The Stl formula ϕ_{dl_free} specifies the deadlock-freedom requirement that if process P_1 attempts to enter the critical section, then there is a trajectory that leads P_1 into its critical section:

$$\forall\Box(pc_1 = reqC \ \rightarrow \ \exists\Diamond(pc_1 = inC)). \qquad (\phi_{dl\_free})$$

Symmetric first-request-first-in and deadlock-freedom requirements can be asserted for process P_2. ■

Exercise 5.5 {} [Equal opportunity] Recall Peterson's mutual-exclusion protocol from Chapter 2. Write an Stl formula ϕ_{bd_ot} that specifies the equal-opportunity requirement that if process P_1 attempts to enter the critical section when process P_2 is in its noncritical section, then P_2 may enter its critical section at most once before P_1 is allowed to enter its critical section. ■

Example 5.5 [Railroad controller] Recall the module RailroadSystem from Chapter 3. The Stl-formula

$$\forall\Box\,\neg(\, pc_W = bridge \ \wedge \ pc_E = bridge\,)$$

specifies the safety requirement that both trains should never be simultaneously on the bridge. The Stl-formula

$$\forall\Box\,\exists\Diamond\,(\, pc_W \neq bridge \ \wedge \ pc_E \neq bridge\,)$$

specifies the requirement that from every reachable state there exists a trajectory leading to a state in which none of the two trains are on the bride. ■

Event Stl

The logic Stl specifies requirements of trajectories using atomic formulas that are interpreted at states. When a module uses events for communication, it is convenient to use formulas that refer to transitions or pairs of states. For this purpose, we define the state logic Estl. Unlike Stl, it has two sorts of formulas: state formulas that are interpreted with respect to states, and transition formulas that are interpreted with respect to transitions.

Event Stl (Estl) is the state logic whose state formulas are generated by the grammar

$$\phi \ ::= \ p \mid \phi \vee \phi \mid \neg\phi \mid \varphi \exists\mathcal{U} \varphi.$$

for atomic formulas p and transition formulas φ. The transition formulas are generated by the grammar

$$\varphi \ ::= \ \phi \mid \bigcirc\phi \mid \varphi \vee \varphi \mid \neg\varphi$$

for state formulas ϕ. The semantics of the state formulas of Estl is defined as follows:

$$\begin{array}{lll}
s \models_K p & \text{iff} & \langle\!\langle s \rangle\!\rangle \models p; \\
s \models_K \phi \vee \psi & \text{iff} & s \models_K \phi \text{ or } s \models_K \psi; \\
s \models_K \neg\phi & \text{iff} & s \not\models_K \phi; \\
s \models_K \rho\exists\mathcal{U}\varphi & \text{iff} & \text{there is a source-}s\text{ trajectory } \overline{s}_{0..m} \text{ of } K \\
 & & \text{with } m > 0 \text{ such that} \\
 & & (1)\ (s_{m-1}, s_m) \models_K \varphi \text{ and} \\
 & & (2) \text{ for all } 0 < i \leq m,\ (s_{i-1}, s_i) \models_K \varphi \vee \rho.
\end{array}$$

The semantics of the transition formulas of Estl is defined as follows:

$$\begin{array}{lll}
(s,t) \models_K \phi & \text{iff} & s \models_K \phi; \\
(s,t) \models_K \bigcirc\phi & \text{iff} & t \models_K \phi; \\
(s,t) \models_K \varphi \vee \rho & \text{iff} & (s,t) \models_K \varphi \text{ or } (s,t) \models_K \rho; \\
(s,t) \models_K \neg\varphi & \text{iff} & (s,t) \not\models_K \varphi.
\end{array}$$

Thus, the transition formulas can refer to the updated values of variables by using the next operator. If x is a boolean variable of the event type, then we use $x?$ as an abbreviation for the transition formula $(x \not\leftrightarrow \bigcirc x)$. Suppose that x and y are two event variables. The following Estl formula asserts that no event x is followed by an event y:

$$\forall\Box(x? \rightarrow \bigcirc\forall\Box\neg y?).$$

Example 5.6 [Synchronous 3-bit Counter] Recall the synchronous module SCountThree of Example 2.19 that models a 3-bit counter. The desired specification of the counter is that in every update round, if the start command is present ($start' = 1$), then the counter should be reset to 0, and if the increment command is present, then the counter should be incremented by 1 (module 8), and otherwise, the counter should stay unchanged. The following Estl formula expresses the desired update of the bit out_0:

$$\forall\Box \left(\begin{array}{lcl} \bigcirc start & \rightarrow & \neg \bigcirc out \\ \neg \bigcirc start \wedge \bigcirc inc & \rightarrow & out \leftrightarrow \neg \bigcirc out \\ \neg \bigcirc start \wedge \neg \bigcirc inc & \rightarrow & out \leftrightarrow \bigcirc out \end{array} \right).$$

The desired update of the remaining two bits can be specified similarly in Estl. ■

Open modules

Checking existential requirements of an open module is not very meaningful. Existential requirements over external variables are trivially satisfied, while existential requirements over interface variables are not preserved under parallel composition.

Remark 5.2 [] Let P be a module, and let p and q be boolean expressions over the external variables of P. Then, for every state s of P, $s \models \exists\bigcirc p$ and $s \models p\exists\mathcal{U}q$. ■

Exercise 5.6 {} [] Give an example of a module $P \,\|\, Q$ and an Stl-formula ϕ such that the answer to the verification problem (P, ϕ) is Yes, while the answer to $(P \,\|\, Q, \phi)$ is No. ■

If we restrict ourselves only to the universal formulas, then the compositionality principle holds. Let ∀Stl be the fragment of Stl generated by the grammar

$$\phi \ ::= \ p \mid \neg p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \forall\bigcirc \phi \mid \phi\forall\mathcal{W}\phi$$

The logic ∀Stl is not closed under negation.

The parallel composition operation on modules ensures that the projection of a trajectory of a compound module onto the variables of a component is a trajectory of the component. This implies that the compositionality principle holds for ∀Stl.

Proposition 5.1 [Compositionality for ∀Stl] If the module P satisfies the ∀Stl-formula ϕ, then for every module Q that is compatible with P, the compound module $P \,\|\, Q$ satisfies ϕ.

Proof. Let P and Q be two compatible modules, and let ϕ be a formula ∀Stl. Let $R = P \,\|\, Q$. We prove that, for every subformula ψ of ϕ, for all states s of P, if $s \models_P \psi$ then for all states t of R, if $t[X_P] = s$ then $t \models_R \psi$. The proof is by

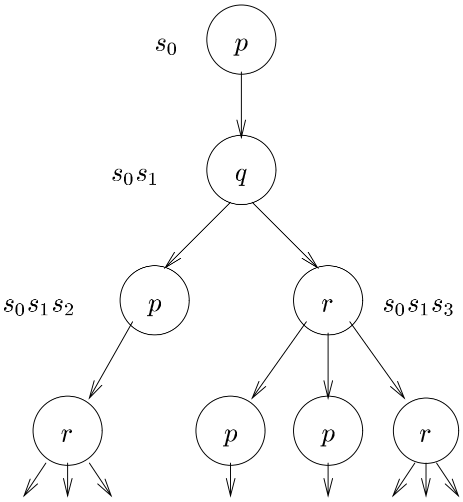

Figure 5.3: Tree Semantics of Stl

induction on the structure of ψ. Consider a state s of P such that $s \models_P \psi$, and let t be a state of R with $t[X_P] = s$. The interesting case is when $\psi = \chi_1 \forall\mathcal{W}\chi_2$.

Consider a source-t trajectory $t_0 \ldots t_m$ of R. Then, from the properties of the parallel composition operation, there exists a trajectory $s_0 \ldots s_m$ of P such that $s_i = t_i[X_P]$ for all $0 \leq i \leq m$. Since $\overline{s}_{0\ldots m}$ is a source-s trajectory of P, and $s \models_P \chi_1 \forall\mathcal{W}\chi_2$, we have either for all $0 \leq i \leq m$, $s_i \models_P \chi_1$; or there is a natural number j with $0 \leq j \leq m$ such that $s_j \models_P \chi_2$ and for all $0 \leq i < j$, $s_i \models_P \chi_1$. From induction hypothesis, it follows that either for all $0 \leq i \leq m$, $t_i \models_R \chi_1$; or there is a natural number j with $0 \leq j \leq m$ such that $t_j \models_R \chi_2$ and for all $0 \leq i < j$, $t_i \models_R \chi_1$. ■

Tree Semantics

The semantics of Stl can, alternatively, be defined using trees. For an observation structure K and a state s of K, the s-rooted tree is obtained by unfolding the source-s trajectories of K into a tree. Figure 5.3 shows the s_0-rooted tree for the transition structure of Figure 5.1.

Formally, for a state s of the observation structure $K = (\Sigma, \sigma^I, \rightarrow, A, \langle\!\langle\cdot\rangle\!\rangle)$ the s-rooted K-tree is another transition structure $T_K(s)$:

- The states of $T_K(s)$ are the source-s trajectories of K.
- The only initial state $T_K(s)$ is its root s.

- There is a transition from $\overline{s}_{0\ldots m}$ to $\overline{t}_{0\ldots n}$ if $n = m + 1$ and $s_i = t_i$ for $0 \leq i \leq m$.
- The observation alphabet is A.
- The observation function maps $\overline{s}_{0\ldots m}$ to $\langle\!\langle s_m \rangle\!\rangle$.

Verify that the structure $T_K(s)$ is a tree, that is, every state, except the root s, has a unique predecessor. The formulas of Stl can be interpreted over the tree $T_K(s)$ instead the structure K:

Proposition 5.2 [Tree property of Stl] Let ϕ be a Stl-formula, let K be a ϕ-structure, and let s be a state of K. For every state $\overline{s}_{0\ldots m}$ of the tree structure $T_K(s)$,

$$\overline{s}_{0\ldots m} \models_{T_K(s)} \phi \text{ iff } s_m \models_K \phi.$$

Exercise 5.7 {} [] Prove Proposition 5.2. ■

Proposition 5.2 implies that the satisfaction nof Stl-formulas at a state s of an observation structure depends only upon the substructure of K that is reachable from s, and is insensitive to the unwinding of the structure.

5.3 Model Checking

We are given an Stl formula ϕ and a ϕ-structure K, and we are asked to check if all initial states of K satisfy ϕ. For this purpose, we find the characteristic region $[\![\phi]\!]_K$, that is, all states of K that satisfy ϕ. We proceed inductively on the structure of the formula ϕ, by first finding the characteristic regions for the subformulas of ϕ.

Subformulas

The set $Sub(\phi)$ of subformulas of the Stl formula ϕ is defined inductively:

$$\begin{aligned}
Sub(p) &= \{p\} \text{ for an atomic formula } p;\\
Sub(\psi \vee \chi) &= \{\psi \vee \chi\} \cup Sub(\psi) \cup Sub(\chi);\\
Sub(\neg\psi) &= \{\neg\psi\} \cup Sub(\psi);\\
Sub(\exists\bigcirc \psi) &= \{\exists\bigcirc \psi\} \cup Sub(\psi);\\
Sub(\psi\exists\mathcal{U}\chi) &= \{\psi\exists\mathcal{U}\chi\} \cup Sub(\psi) \cup Sub(\chi).
\end{aligned}$$

Remark 5.3 [Number of subformulas] The Stl formula ϕ has at most $|\phi|$ subformulas. ■

Given the input formula ϕ, the model-checking algorithm for Stl calls a function $\mathit{OrderedSub}(\phi)$, which returns a queue with the subformulas of ϕ such that a formula appears only after all its subformulas. Assuming a type **form** for formulas:

> function $\mathit{OrderedSub}$: queue of form
>
> Input: an Stl formula ϕ.
> Output: a queue of the formulas in $\mathit{Sub}(\phi)$ such that if $\psi \in \mathit{Sub}(\phi)$ and $\chi \in \mathit{Sub}(\psi)$, then χ precedes ψ in $\mathit{OrderedSub}(\phi)$.

Example 5.7 [Subformula ordering] For example, the function call $\mathit{OrderedSub}((p \wedge q) \exists\mathcal{U} (\neg \exists \bigcirc r))$ may return the queue

$$(p, q, r, p \wedge q, \exists \bigcirc r, \neg \exists \bigcirc r, ((p \wedge q) \exists\mathcal{U} (\neg \exists \bigcirc r))$$

of formulas. ■

Exercise 5.8 {} [Computing subformulas] Give an algorithm that, given an Stl-formula ϕ with ℓ symbols, computes the function $\mathit{OrderedSub}(\phi)$ in $O(\ell)$ time. ■

5.3.1 Enumerative Stl model checking

An enumerative model-checking algorithm computes an enumerative representation of the characteristic region $[\![\phi]\!]_K$ from enumerative representations of the characteristic regions for the subformulas of ϕ.

Assume that the given observation structure K is finite. An enumerative Stl model-checking algorithm computes, for each state s of K, the set $\lambda(s) \subseteq \mathit{Sub}(\phi)$ of subformulas of ϕ that are satisfied by s. Initially, $\lambda(s)$ is empty for each state s. Then, all subformulas of ϕ are considered in the order given by the function call $\mathit{OrderedSub}(\phi)$. Consider a subformula ψ of ϕ. For each state s of K, we must decide whether s satisfies ψ, and update $\lambda(s)$ accordingly. Inductively, we know that for every subformula χ of ψ and for each state s, the formula χ belongs to $\lambda(s)$ iff $s \models_K \chi$. The form of ψ leads to various cases. The interesting case occurs when ψ has the form $\chi_1 \exists\mathcal{U} \chi_2$. In this case, we define a finite transition graph H:

> The vertices of H are the states Σ of K. A state $s \in \Sigma$ is an initial state of H iff the formula χ_2 belongs to $\lambda(s)$. The graph H has an edge from $s \in \Sigma$ to $t \in \Sigma$ iff (1) K has a transition from t to s and (2) the formula χ_1 belongs to $\lambda(t)$.

The semantics of the possibly-until connective implies that $s \models_K \chi_1 \exists\mathcal{U} \chi_2$ iff the vertex s is reachable in the graph H. Consequently, the set of states that satisfy ψ can be computed by a depth-first search in H.

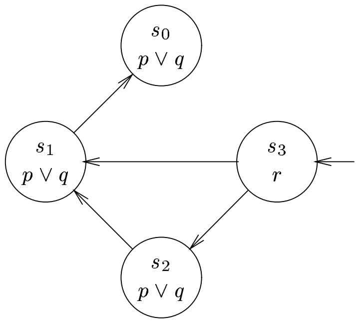

Figure 5.4: Reachability analysis for checking possibly-until

Example 5.8 [] Consider the observation structure of Figure 5.1, and the possibly-until formula $\phi = (p \vee q)\exists\mathcal{U}r$. Let $OrderedSub(\phi) = \{p, q, r, p \vee q, \phi\}$. First, the formula p is added to the sets $\lambda(s_0)$ and $\lambda(s_2)$. Then, the formula q is added to the set $\lambda(s_1)$. Then, the formula r is added to the set $\lambda(s_3)$. Then, the formula $p \vee q$ is added to the sets $\lambda(s_0)$, $\lambda(s_1)$, and $\lambda(s_2)$. To evaluate the truth of ϕ, consider the transition graph of Figure 5.4 with the initial state s_3. All the states are reachable, implying that the formula ϕ is satisfied in all the states, and hence, for every state s, ϕ is added to the set $\lambda(s)$. ■

The Stl model-checking algorithm shown in Figure 5.5 considers only the reachable substructure of the input structure K. For this purpose, the algorithm calls the function *Reach*(K), which returns a queue with the reachable states of K. The function *Reach* can be implemented using the techniques from Chapter 3. The abstract type for the input structure K supports also the operations *InitQueue*, *PreQueue*, and *PostQueue* (see Chapter 3). Given a state s of K, each function call $\lambda(s)$ returns a set of formulas. The abstract type set for the formula sets $\lambda(s)$: **set of form** and the state set σ: **set of state** supports the operations *EmptySet*, *Insert*, and *IsMember*. The satisfaction of atomic formulas is checked by the function AtomicCheck. For an observation structure K, a state s of K, and an atomic formula p, *AtomicCheck*(s, p, K) returns true iff $s \models_K p$. Checking of the possibly-until formulas employs a depth-first search using the stack τ and the set σ that stores the states visited by the search.

Lemma 5.1 [Correctness of enumerative model checking] Let ϕ be an Stl formula, and let K be a ϕ-structure with a finite reachable substructure. Upon termination of Algorithm 5.1, for every subformula ψ of ϕ and each state s of K, $\psi \in \lambda(s)$ iff $s \models_K \psi$.

Exercise 5.9 {} [] Consider the module Pete from Chapter 2, and the Stl-formula ϕ_{fifo} of Example 5.4. Execute Algorithm 5.1 on the input (K_{Pete}, ϕ_{fifo}), and establish that the answer to the verification problem $(Pete, \phi_{fifo})$ is Yes. ■

Algorithm 5.1 [Enumerative Stl Model Checking]

Input: an Stl formula ϕ, and a ϕ-structure K whose reachable substructure is finite.
Output: the answer to the model-checking problem (K, ϕ).

```
σ^R := Reach(K);
foreach s in σ^R do λ(s) := EmptySet od;
foreach ψ in OrderedSub(φ) do
  case ψ = p for an atomic formula p:
    foreach s in σ^R do
      if AtomicCheck(s, p, K) then λ(s) := Insert(p, λ(s)) fi
      od
  case ψ = χ1 ∨ χ2:
    foreach s in σ^R do
      if IsMember(χ1, λ(s)) or IsMember(χ2, λ(s)) then
        λ(s) := Insert(ψ, λ(s)) fi
      od
  case ψ = ¬χ:
    foreach s in σ^R do
      if not IsMember(χ, λ(s)) then λ(s) := Insert(ψ, λ(s)) fi
      od
  case ψ = ∃○ χ:
    foreach s in σ^R do
      foreach t in PostQueue(s, K) do
        if IsMember(χ, λ(t)) then λ(s) := Insert(ψ, λ(s)) fi
        od
      od
  case ψ = χ1∃𝒰χ2:
    σ := EmptySet; τ := EmptyStack;
    foreach s in σ^R do
      if IsMember(χ2, λ(s)) and not IsMember(s, σ) then
        τ := Push(s, τ); σ := Insert(s, σ) fi;
      while not EmptySet(τ) do
        t := Top(τ); τ := Pop(τ); λ(t) := Insert(ψ, λ(t));
        foreach u in PreQueue(t, K) do
          if IsMember(χ1, λ(u)) and not IsMember(u, σ) then
            τ := Push(u, τ); σ := Insert(u, σ) fi
          od
        od
      od
    end case
  od;
foreach s in InitQueue(K) do
  if not IsMember(φ, λ(s)) then return No fi
  od;
return Yes.
```

Figure 5.5: Enumerative Stl model checking

If the observation structure K is finite, then K can be represented by a record $\{K\}_e$ with two components of type enumgraph and **array**[**state**] **of obs**, where obs is the type of observations. The second component is redundant if the observation structure K is defined by a reactive module. In this case, the observation of each state can be obtained from the state itself by ignoring the values of the private variables. The enumerative structure representation $\{K\}_e$ supports the operations *InitQueue*, *PreQueue*, and *PostQueue* in constant time, and *Reach* can be implemented in time proportional to the number of transitions of K (see Chapter 3). The function λ can be implemented in constant time using an array $\lambda :$ **array**[**state**] **of set of form**, where form ranges over the subformulas of ϕ. If the abstract type **set of** $\mathbb{T}$ is represented by a boolean array **array**[$\mathbb{T}$] **of** $\mathbb{B}$, then the operations *Insert* and *IsMember* require constant time, and the operation *EmptySet* requires time proportional to the number of elements in $\mathbb{T}$. This representation leads to linear running time of Algorithm 5.1.

Theorem 5.1 [Stl model checking] Let ϕ be an Stl formula with ℓ symbols, and let K be a finite ϕ-structure with n states and m transitions. Suppose that every call to the function AtomicCheck requires constant time. Given the input ϕ and $\{K\}_e$, Algorithm 5.1 solves the model-checking problem (K, ϕ) in $O(\ell \cdot (n + m))$ time.

Remark 5.4 [Space complexity of Stl model checking] Let ϕ be an Stl formula with ℓ symbols, and let K be a finite ϕ-structure with n states. Algorithm 5.1 requires $O(\ell \cdot n)$ space. It is possible to solve the model checking problem in recursively top-down manner to save space. In particular, there is a nondeterministic algorithm that requires space $O(\ell \cdot \log\ n)$ space. If the Stl-formula ϕ is small, that is, bounded by a constant, then the complexity class of the model checking problem (K, ϕ) is NLOGSPACE. ■

The Stl-verification problem (P, ϕ), for a finite module P and an Stl formula ϕ, can be solved by first constructing the enumerative structure representation $\{K_P\}_e$, and then applying Algorithm 5.1. Since the number of states of K_P may be exponentially larger than the description of P, the resulting cost for Stl verification is exponential. This cost is unavoidable, because already the propositional Stl-verification problem is PSPACE-hard (the propositional invariant-verification problem (P, p) and the Stl-verification problem $(P, \forall\Box p)$ have the same answer, and the former was shown to be PSPACE-hard in Chapter 3). In Section 3.2.4, we considered two space optimizations for invariant verification, using on-the-fly representations and using only the latched variables. Both these techniques are useful for improving efficiency of Stl model checking.

Exercise 5.10 {} [Stl verification in PSPACE] Prove that the Stl-verification problem is in PSPACE. ■

Algorithm 5.2 [Symbolic Stl model checking]

Input: an Stl formula ϕ, and a ϕ-structure K.
Output: the answer to the model-checking problem (K, ϕ).

foreach ψ in $\mathit{OrderedSub}(\phi)$ do
  case $\psi = p$ for an atomic formula p: $[\![\psi]\!] = [\![p]\!]$
  case $\psi = \chi_1 \vee \chi_2$: $[\![\psi]\!] = [\![\chi_1]\!] \cup [\![\chi_2]\!]$
  case $\psi = \neg\chi$: $[\![\psi]\!] = [\![\mathit{true}]\!] \backslash [\![\chi]\!]$
  case $\psi = \exists\bigcirc\chi$: $[\![\psi]\!] = \mathit{PreReg}([\![\chi]\!], K)$
  case $\psi = \chi_1 \exists\mathcal{U} \chi_2$:
    $\sigma := [\![\mathit{false}]\!]$;
    $\tau := [\![\chi_2]\!]$;
    while $\tau \not\subseteq \sigma$ do
      $\sigma := \sigma \cup \tau$;
      $\tau := \mathit{PreReg}(\sigma, K) \cap [\![\chi_1]\!]$
      od
  end case
  od;
if $\mathit{InitReg}(K) \subseteq [\![\phi]\!]$ then return Yes else return No.

Figure 5.6: Symbolic Stl model checking

Exercise 5.11 {} [Reduced observation structure] Recall the definition of the reduced transition graph of a module. Define the reduced observation structure of a module and use it for an improved solution of the Stl-verification problem, along the lines of Theorem 3.4. ■

5.3.2 Symbolic Stl model checking

A symbolic model-checking algorithm computes a symbolic representation of the characteristic region $[\![\phi]\!]_K$ from symbolic representations of the characteristic regions for the subformulas of ϕ. The symbolic Stl model-checking algorithm shown in Figure 5.6 assumes that the symbolic structure representation supports the operations *InitReg* and *PreReg*, and the symbolic region representation supports, in addition to the operations $\cup$, $\cap$, and $\subseteq$, also the set difference operation $\backslash$. An Stl-verification problem (P, ϕ) can be solved by first constructing a symbolic representation of the observation structure K_P, and then applying Algorithm 5.2.

Consider the propositional Stl-verification problem (P, ϕ). The symbolic representation of the observation structure K_P consists of (1) the symbolic representation of the transition graph G_P (which consists of the initial predicate

q^I over X_P and the transition predicate q^T over $X_P \cup X'_P$), and (2) the set $\mathsf{obs}X_P$ of observable variables. Binary decision diagrams are suitable for solving the propositional Stl-verification problem. All the heuristics considered in Section 5.2.4 to improve the efficiency of BDD-based representations are useful in Stl-verification.

Exercise 5.12 {} [Symbolic region difference] Write an algorithm that computes, given the BDD representation of two propositional formulas p and q, the BDD representation of the difference $[\![p]\!]\backslash[\![q]\!]$. What is time complexity of the algorithm? ■

Exercise 5.13 {} [Event Stl] Write a symbolic model-checking algorithm for Estl that computes characteristic regions only over the latched variables. ■

5.4 The Distinguishing Power of Stl

The partition-refinement algorithms presented in Chapter 6 can be used to reduce the size of an observation structure, while retaining the ability of computing the characteristic regions for Stl formulas. For this purpose, we need to understand when two states of an observation structure satisfy the same Stl formulas.

5.4.1 State Equivalences

A state equivalence $\simeq$ is a family of relations which contains for each observation structure K an equivalence $\simeq_K$ on the state space of K (that is, a partition of the state-space of K). Here are three examples of state equivalences:

1. Within each observation structure K, state equality $=$ distinguishes any two different states: for every state s of K, $s/_{=_K}$ is the singleton set $\{s\}$.
2. Within each observation structure K, observational equivalence $\approx$ distinguishes any two states with different observations: if $\langle\!\langle\cdot\rangle\!\rangle$ is the observation function of K, then two states s and t of K are observationally equivalent, denoted $s \approx_K t$, if $\langle\!\langle s\rangle\!\rangle = \langle\!\langle t\rangle\!\rangle$.
3. Within each observation structure K, universal equivalence $\simeq^U$ does not distinguish any two states: if Σ is the state space of K, then for every state s of K, $s/_{\simeq^U}$ equals Σ.

The refinement relation on equivalences induces a preorder on state equivalences. Let $\simeq^1$ and $\simeq^2$ be two state equivalences. The state equivalence $\simeq^1$ is as distinguishing as the state equivalence $\simeq^2$, written $\simeq^2 \sqsubseteq \simeq^1$, if for all observation structures K, the equivalence $\simeq^1_K$ refines the equivalence $\simeq^2_K$. The state equivalence $\simeq^1$ is more distinguishing than $\simeq^2$, if $\simeq^2 \sqsubseteq \simeq^1$ and $\simeq^1 \not\sqsubseteq \simeq^2$. The two state equivalences $\simeq^1$ and $\simeq^2$ are equally distinguishing, if $\simeq^1 \sqsubseteq \simeq^2$ and $\simeq^2 \sqsubseteq \simeq^1$. The two state equivalences $\simeq^1$ and $\simeq^2$ are incomparable if $\simeq^1 \not\sqsubseteq \simeq^2$ and $\simeq^2 \not\sqsubseteq \simeq^1$.

Remark 5.5 [State equality and universal equivalence] Let $\simeq$ be a state equivalence. The state equivalence $\simeq$ is as distinguishing as universal equivalence, and state equality is as distinguishing as $\simeq$. In other words, the preorder $\sqsubseteq$ has a bottom, the universal equivalence $\simeq^U$, and a top, the state equality $=$. ■

5.4.2 Bisimilarity

Since observational equivalence refers to static observation snapshots for distinguishing two states, more distinctions can be made by referring also to dynamic observation sequences. Bisimilarity is such a state equivalence which, while less distinguishing than state equality, is more distinguishing than observational equivalence.

> Bisimilarity
> Let K be an observation structure. The coarsest stable refinement $\simeq_K^B = \min_K(\approx_K)$ of observational equivalence and the induced state equivalence $\simeq^B$ are called bisimilarity.

Remark 5.6 [Alternative definition of bisimilarity] Let $K = (\Sigma, \sigma^I, \rightarrow, A, \langle\!\langle\cdot\rangle\!\rangle)$ be an observation structure. The equivalence $\cong$ on the states of K is a bisimulation of K if (1) the partition $\Sigma/_{\cong}$ is a stable partition of K and (2) $\cong$ refines the observational equivalence $\approx_K$. Thus, for all states s and t of K, if $s \cong t$ then

(1) $\langle\!\langle s\rangle\!\rangle = \langle\!\langle t\rangle\!\rangle$;
(2) if $s \rightarrow s'$, then there is a state t' such that $t \rightarrow t'$ and $s' \cong t'$;
(3) if $t \rightarrow t'$, then there is a state s' such that $s \rightarrow s'$ and $s' \cong t'$.

Two states s and t of K are bisimilar iff there is a bisimulation $\cong$ of K such that $s \cong t$. ■

It follows that bisimilarity can be characterized game-theoretically. Consider the following two-player game on the graph of the observation structure K. Player I, the protagonist, attempts to show that two given states s and t are bisimilar, while Player II, the adversary, tries to distinguish the two states. If the two given states have different observations, then the adversary wins immediately. Throughout the game, there are two active states; initially s and t are active. Each move of the game consists of two parts —a move by the adversary followed by a move of the protagonist. The adversary picks one of the two active states and replaces it by one of its successors, say s'; the protagonist, then, must match

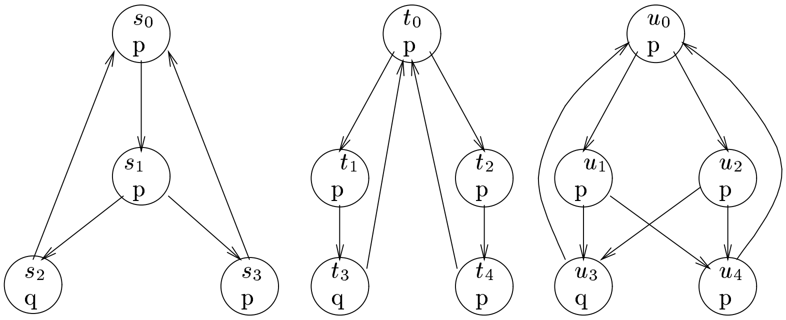

Figure 5.7: Bisimilarity game

the move of the adversary by replacing the other active state with one of its successors t' such that $\langle\langle s' \rangle\rangle = \langle\langle t' \rangle\rangle$. If the protagonist cannot match a move of the adversary, then the adversary wins the game. The two initial states s and t are bisimilar iff the adversary does not have a winning strategy; that is, all of possible moves of the adversary can perpetually be matched by the protagonist.

Example 5.9 [Bisimilarity game] Consider the observation structure shown in Figure 5.7. The two states s_0 and t_0 are not bisimilar. To see this using the bisimilarity game, consider the following strategy for the adversary. The adversary chooses s_0 and moves to s_1. If, in response, the protagonist decides to move from t_0 to t_1, then the adversary moves from s_1 to s_3, and the protagonist cannot match this move (because no transition from t_1 leads to a state with observation p). Similarly, if the protagonist decides to move from t_0 to t_2, then the adversary moves from s_1 to s_2, and the protagonist cannot match this move either. So the adversary has a winning strategy in the bisimilarity game, which implies that the two states s_0 and t_0 are not bisimilar. By contrast, it is easy to check that the two states s_0 and u_0 are bisimilar. ■

Exercise 5.14 {} [Fixpoint view of bisimilarity] Let $K = (\Sigma, \sigma^I, \rightarrow, A, \langle\langle \cdot \rangle\rangle)$ be an observation structure. Given a binary relation $\cong \subseteq \Sigma^2$, we define the binary relation $f(\cong) \subseteq \Sigma^2$ such that for all states s and t of K, $(s, t) \in f(\cong)$ iff

(1) $s \cong t$;
(2) if $s \rightarrow s'$, then there is a state t' such that $t \rightarrow t'$ and $s' \cong t'$;
(3) if $t \rightarrow t'$, then there is a state s' such that $s \rightarrow s'$ and $s' \cong t'$.

First, prove that f is a monotonic function on the complete partial order of the equivalences on Σ under refinement (i.e., if $\cong_1 \preceq \cong_2$ for two equivalences $\cong_1$ and $\cong_2$ on Σ, then $f(\cong_1) \preceq f(\cong_2)$). Second, prove the binary relation $\cong \subseteq \Sigma^2$ is a bisimulation of K iff (1) $\cong$ is an equivalence that refines the observational equivalence $\approx$ and (2) $\cong$ is a fixpoint of f. Third, conclude that bisimilarity $\simeq_K^B$ is the greatest fixpoint of f. ■

An infinite hierarchy of state equivalences

Bisimilarity —a family of coarsest stable refinements— is defined to be the least distinguishing state equivalence in the set of state equivalences whose constituents are stable refinements of observational equivalence. Hence, bisimilarity is defined "from below," in terms of more distinguishing state equivalences. Alternatively, bisimilarity can be defined "from above," as the limit of a sequence of less distinguishing state equivalences. Intuitively, two states are i-step bisimilar, written $\approx^i$ for a natural number i, if in the bisimilarity game the adversary has no winning strategy that requires at most i moves. Then, two states are bisimilar if they are i-step bisimilar for all natural numbers i.

i-step bisimilarity

The state equivalences $\approx^i$, called i-step bisimilarity for each natural number i, are defined inductively. The state equivalence $\approx^0$ coincides with observational equivalence; that is, $\approx^0 = \approx$. For each natural number i, for every observation structure $K = (\Sigma, \sigma^I, \rightarrow, A, \langle\langle\cdot\rangle\rangle)$, and for all states s and t of K, let $s \approx_K^{i+1} t$ iff

(1) $\langle\langle s\rangle\rangle = \langle\langle t\rangle\rangle$;
(2) if $s \rightarrow s'$, then there is a state t' such that $t \rightarrow t'$ and $s' \approx_K^i t'$;
(3) if $t \rightarrow t'$, then there is a state s' such that $s \rightarrow s'$ and $s' \approx_K^i t'$.

Remark 5.7 [] For each natural number i, the state equivalence $\approx^{i+1}$ is as distinguishing as $\approx^i$, and bisimilarity is as distinguishing as $\approx^i$. ■

Exercise 5.15 {} [Computation of i-step bisimilarity] Write an algorithm that, given an observation structure K and a natural number i, computes the partition $K/_{\approx_K^i}$. What is the asymptotic running time of your algorithm for finite input structures? ■

The hierarchy of state equivalences given by i-step bisimilarity is strict and converges towards bisimilarity.

Proposition 5.3 [Strict hierarchy] For each natural number i, the state equivalence $\approx^{i+1}$ is more distinguishing than the state equivalence $\approx^i$.

Proof. The proof is by induction on i. We only give the base case and indicate the idea behind the inductive step.

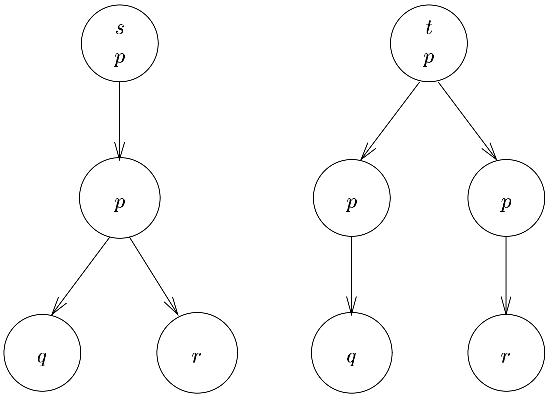

Figure 5.8: The observation structure K_1

For the base case, consider the observation structure K_1 shown in Figure 5.8. To distinguish the two states s and t in the bisimilarity game, the adversary needs a single move (the adversary chooses the successor of s). It follows that $s \approx^0_{K_1} t$ and $s \not\approx^1_{K_1} t$; that is, $\approx^1$ is more distinguishing than $\approx^0$.

For the idea behind the inductive step, consider the observation structure K_2 shown in Figure 5.9. To distinguish the two states s and t, the adversary needs two moves (with its first move, the adversary chooses the left successor of s; with its second move, it chooses the rightmost successor of s'). It follows that $s \approx^1_{K_2} t$ and $s \not\approx^2_{K_2} t$; that is, $\approx^2$ is more distinguishing than $\approx^1$. ■

Exercise 5.16 {} [] Give a complete proof of Proposition 5.3. Use only finite observation structures to distinguish $\approx^{i+1}$ from $\approx^i$. ■

The next proposition gives establishes that bisimilarity coincides with the inductive definition.

Proposition 5.4 [Alternative definition of bisimilarity] Bisimilarity $\simeq^B$ equals the intersection $\bigcup i \in \mathbb{N}.\ \approx^i$ of the i-step bisimilarity equivalences.

Proof. We show that the function f from Exercise 5.14 is $\bigcap$-continuous; that is, given an observation structure $K = (\Sigma, \sigma^I, \rightarrow, A, \langle\langle\cdot\rangle\rangle)$ and a sequence $\cong_0, \cong_1, \cong_2, \ldots$ of equivalences on Σ, if $\approx \succeq \cong_0 \succeq \cong_1 \succeq \cong_2 \succeq \cdots$, then $f(\cap i \in \mathbb{N}.\ \cong_i)$ equals $\cap i \in \mathbb{N}. f(\cong_i)$. The proposition follows from the Kleene fixpoint theorem.

By the monotonicity of f (Exercise 5.14), $f(\cap i \in \mathbb{N}.\ \cong_i) \preceq (\cap i \in \mathbb{N}. f(\cong_i))$. Conversely, consider two states s and t of K such that for all natural numbers i, $(s,t) \in f(\cong_i)$. Let $s \rightarrow s'$. Then for all natural numbers i, there is a state t_i with $t \rightarrow t_i$ such that $s' \cong_i t_i$. Since all the equivalences $\cong_i$ refine the

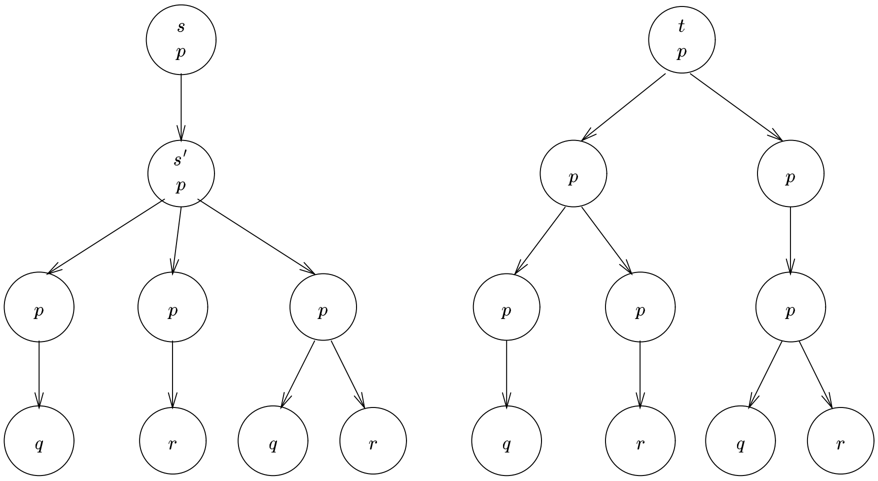

Figure 5.9: The observation structure K_2

propositional equivalence $\approx$, $s' \approx t_i$ for all i. That is, all the state t_i have identical observations. Since $\langle\!\langle post(s)\rangle\!\rangle$ is finite, there is a state t' with $t \to t'$ such that $s' \cong_i t'$ for infinitely many natural numbers i. Since $i \leq j$ implies $\cong_i \succeq \cong_j$, $s' \cong_i t'$ for all natural numbers i. Therefore, $(s,t) \in f(\cap i \in \mathbb{N}.\ \cong_i)$. ■

Remark 5.8 [] The requirement that every state has only finitely many successors per observation is essential for the validity of Proposition 5.4. Otherwise the function f from Exercise 5.14 is not necessarily $\bigcap$-continuous, and $\simeq_K^B$ properly refines $\bigcup i \in \mathbb{N}.\ \approx^i$. ■

5.4.3 Requirement-preserving Equivalences

Every state logic induces a state equivalence, namely, the state equivalence that distinguishes any two states iff there is a state formula that is satisfied by one state but not by the other state.

> Φ-equivalence
>
> Let Φ be a state logic and let K be an observation structure. The two states s and t of K are Φ-equivalent, denoted $s \simeq_K^\Phi t$, if for all Φ-formulas ϕ such that K is a ϕ-structure, $s \models_K \phi$ iff $t \models_K \phi$. The equivalence $\simeq_K^\Phi$ and the induced state equivalence $\simeq^\Phi$ are called Φ-equivalence.

The state equivalences induced by state logics allow us to compare the distinguishing power of two state logics. Let Φ and Ψ be two state logics. The state logic Φ is as distinguishing as the state logic Ψ if $\simeq^\Psi\ \sqsubseteq\ \simeq^\Phi$, etc. In other words, the state logic Φ is as distinguishing as the state logic Ψ, if whenever some Ψ-formula distinguishes between two states, there exists some Φ-formula that distinguishes between those two states: for every observation structure K, and for every two states s and t of K, if there exists a Ψ-formula ϕ such that $s \models_K \phi$ but $t \not\models_K \phi$, then there there exists a Φ-formula ψ such that $s \models_K \psi$ but $t \not\models_K \psi$.

Abstraction

Consider the model checking problem (K, ϕ) for a state logic Φ. The notion of abstraction defines the conditions under which computing the characteristic region of K for ϕ can be reduced to computing the characteristic region of a quotient structure of K.

> Abstraction
>
> The state logic Φ admits abstraction if for every state equivalence $\simeq$ as distinguishing as $\simeq^\Phi$, for every Φ-formula ϕ, and for every ϕ-structure K, the characteristic region $[\![\phi]\!]_K$ is $\bigcup [\![\phi]\!]_{K/\simeq}$. If Φ admits abstraction and $\simeq^\Phi$ $\sqsubseteq$ $\simeq$, then $\simeq$ is called an abstract semantics for Φ; if Φ admits abstraction, then $\simeq^\Phi$ is the fully abstract semantics for Φ.

Let Φ be a state logic, let ϕ be a Φ-formula, and let K be a ϕ-structure. Suppose $\simeq$ is an abstract semantics for Φ. Then, any two states that are $\simeq$-equivalent satisfy the same set of Φ-formulas. Hence, instead of performing model checking on the structure K, we can perform model checking on the quotient structure $K/_{\simeq_K}$. Since the logic Φ admist abstraction, we know that a state s of K satisfies a Φ-formula ϕ iff the $\simeq$-equivalence class containing s satisfies ϕ in the quotient structure. Thus, the model-checking problems (K, ϕ) and $(K/_{\simeq_K}, \phi)$ have the same answer. The latter problem may be much simpler, because the state space of the quotient structure $K/_{\simeq_K}$ may be much smaller than the state space of K.

All the state logics that we study, including the logic Stl, admit abstraction. However, it is possible to define operators whose truth is not preserved by quotients.

Proposition 5.5 [Stl abstraction] Stl admits abstraction.

Exercise 5.17 {} [] Prove Proposition 5.5. ■

Example 5.10 [] Consider the state logic Φ^{na} whose state formulas are generated by the grammar

$$\phi \ ::= \ p \mid \phi \vee \phi \mid \neg\phi \mid [even]\phi.$$

The semantics of the operator $[even]$ is defined by the clause

$$s \models_K [even]\phi \quad \text{iff} \quad \text{the characteristic region } [\![\phi]\!]_K \text{ has even cardinality.}$$

For instance, $[even]p$ holds in a state of K iff even number of states of K satisfy p. The logic Φ^{na} does not admit abstraction. Verify that the observational equivalence $\approx$ is the Φ^{na}-equivalence, i.e. no Φ^{na}-formula distinguishes between two states with identical observations. Consider an observation structure K with states Σ and observations A. The characteristic region $[\![[even]true]\!]_K$ equals Σ if Σ has even cardinality and $\emptyset$ otherwise. The quotient $K/_\approx$ has one state per observation of K. Consequently, $\bigcup[\![[even]true]\!]_{K/_\approx}$ equals Σ if A has even cardinality and $\emptyset$ otherwise. Since $|\Sigma|$ may be even while $|A|$ is not, Φ^{na} does not admit abstraction. Consequently, we cannot use quotients to solve the model checking problem for Φ^{na}. ■

Stl equivalence

To use Proposition 5.5 for Stl model checking, we need to determine abstract semantics for Stl. The next proposition asserts that no Stl-formula can distinguish between two bisimilar states.

Proposition 5.6 [Stl abstraction] Bisimilarity is an abstract semantics for Stl.

Proof. Consider an observation structure K. The bisimilarity partition $\simeq^B_K$ is the coarsest stable refinement $\min(\approx_K)$ of the observational equivalence. We need to prove that two bisimilar states satisfy the same set of Stl-formulas. Let ϕ be a formula of Stl. The proof is by induction on the structure of ϕ. Consider two states s and t such that $s \simeq^B t$. We want to prove that $s \models \phi$ iff $t \models \phi$.

The base case is when ϕ is an observation predicate. Since $\simeq^B$ refines the observational equivalence, we know that s and t have identical observations. Hence, s and t satisfy the same set of observation predicates. The inductive case corresponding to logical connectives is straightforward.

Consider the case $\phi = \psi \exists \mathcal{U} \chi$. By inductive hypothesis, bisimilar states agree on the truth of ψ and χ. Suppose $s \models \phi$. Then, there is a source-s trajectory $\overline{s}_{0..m}$ such that $s_m \models \chi$, and $s_i \models \psi$ for $0 \leq i < m$. Since the partition $\simeq^B$ is stable, starting with state $t_0 = t$, we can find states $t_1, t_2, \ldots, t_m$ such that each $t_i \simeq^B s_i$ for $0 \leq i \leq m$, and $t_0 t_1 \ldots t_m$ is a trajectory of K. From the inductive hypothesis, $t_m \models \chi$ and $t_i \models \psi$ for $0 \leq i < m$. This implies that $t \models \phi$. From symmetry, $s \models \phi$ iff $t \models \phi$.

The remaining case $\phi = \exists \bigcirc \psi$ is left for the reader to verify. ■

This suggests that for Stl model checking, it suffices to construct quotients with respect to bisimilarity. Given an observation structure $K = (\Sigma, \sigma^I, \rightarrow, A, \langle\!\langle\cdot\rangle\!\rangle)$, we first consider the observational equivalence $\approx_K$ over the states Σ. The next step is to construct the coarsest stable refinement $\min(\approx_K)$ using one of the algorithms from Chapter 6. This yields the bisimilarity equivalence $\simeq_K^B$ and the minimal quotient $K/_{\simeq^B}$. Then, Stl specifications for the observation structure K can be checked by model checking over the quotient structure $K/_{\simeq^B}$.

We proceed to establish that bisimilarity is a fully abstract semantics of Stl. In fact, if bisimilarity distinguishes two states of an observation structure, then the two states can be distinguished by a Stl formula that employs only the next connective. For instance, in Figure 5.7, the nonbisimilar states s and t can be distinguished by the Stl formula $\exists\bigcirc(\exists\bigcirc q \wedge \exists\bigcirc r)$. The fragment Stl$^{\bigcirc}$ of Stl contains those formulas of Stl that do not contain the until connective $\exists\mathcal{U}$. The fragment Stl$^{\mathcal{U}}$ of Stl contains those formulas of Stl that do not contain the next connective $\exists\bigcirc$. Then Stl is as distinguishing as Stl$^{\bigcirc}$, and also as distinguishing as Stl$^{\mathcal{U}}$.

Proposition 5.7 [Stl full abstraction] The equivalence induced by Stl$^{\bigcirc}$ coincides with the bisimilarity $\simeq^B$.

Proof. Consider an observation structure K. We wish to prove that whenever two states s and t of K belong to different equivalence classes of $\simeq_K^B$, there exists a formula ϕ of Stl$^{\bigcirc}$ such that $s \models_K \phi$ and $t \not\models_K \phi$. We prove that, for every natural number i, for every equivalence class σ of the i-step bisimilarity $\approx_K^i$, there exists a formula ϕ_σ of Stl$^{\bigcirc}$ such that $[\![\phi_\sigma]\!]_K = \sigma$.

Base case $i = 0$: for an equivalence class σ of the propositional equivalence $\approx_K$, the formula ϕ_σ is the observation of σ.

Inductive case $i = k+1$: Let σ be an equivalence class of $\approx_K^{k+1}$. There are only finitely many equivalence classes τ of the partition $\approx_K^k$ such that $s \to t$ for some $s \in \sigma$ and $t \in \tau$. Then, choose

$$\phi_\sigma \;=\; \bigwedge_{\{\tau\in\approx^k \mid \sigma\to\tau\}} \exists\bigcirc\, \phi_\tau \;\wedge\; \forall\bigcirc \bigvee_{\{\tau\in\approx^k \mid \sigma\to\tau\}} \phi_\tau .$$

The reader should verify that the characteristic region $[\![\phi_\sigma]\!]_K$ equals σ. ■

Exercise 5.18 {} [Event Stl] Does Estl admit abstraction? What is the state equivalence induced by Estl? Prove your answers. ■

5.4.4 Stutter-insensitive Equivalences

A reactive module stutters when its observable state stays unchanged. An asynchronous module may stutter in every update round. If the number of rounds for which a module stutters before updating its observation is irrelevant, then many such rounds can be combined into a single transition. This suggests defining a closure operation on observation structures that adds a transition from the state s to the state t whenever there is a trajectory from s to t along which the observation stays unchanged.

Stutter closure

Let $K = (\Sigma, \sigma^I, \rightarrow, A, \langle\!\langle\cdot\rangle\!\rangle)$ be an observation structure. For two states s and t of K, let $s \rightarrow^S t$ if there is an source-s K-trajectory $\overline{s}_{0..m}$ such that (1) for all $0 \leq i < m$, $\langle\!\langle s_i \rangle\!\rangle = \langle\!\langle s \rangle\!\rangle$, and (2) $s_m = t$. The relation $\rightarrow^S$ is called the stutter-closed transition relation of K. The stutter closure K^S is the observation structure $(\Sigma, \sigma^I, \rightarrow^S, A, \langle\!\langle\cdot\rangle\!\rangle)$.

Remark 5.9 [] The stutter-closed transition relation is reflexive. The reachable region of the stutter closure of the observation structure K coincides with the reachable region of K. ■

Exercise 5.19 {} [Stutter closure] Let K be a finite observation structure with n states. Give an $O(n^3)$ algorithm that computes the stutter closure K^S. ■

Stutter closure operation extends to state equivalences also. For instance, two check whether states s and t of an observation structure K are equivalent according to the stutter-closure of bisimilarity, we first compute the stutter-closure of K and then check if the two states s and t are bisimilar.

Stutter closure of state equivalences

Let $\simeq$ be a state equivalence, and let K be an observation structure. For two states s and t of K, $s \cong_K t$, for the stutter closure $\cong$ of $\simeq$, if $s \simeq_{K^S} t$. The induced state equivalence $\cong$ is called the stutter closure of $\simeq$. The state equivalence $\simeq$ is stutter-insensitive if $\simeq = \cong$.

Thus, the equivalence $\cong_K$ is same as the equivalence $\simeq_{K^S}$. For instance, for the structure $K_\bigcirc$ of Figure 5.11 and the bisimilarity partition $\Sigma/_{\simeq^B}$ containing the three singleton regions $\{s\}$, $\{t\}$, and $\{u\}$, the stutter closure $\Sigma/_{\cong^B}$ contains the two regions $\{s,t\}$ and $\{u\}$.

Remark 5.10 [] Observational equivalence is stutter-insensitive, and bisimilarity is not. ■

Proposition 5.8 [Stutter closure] Bisimilarity $\simeq^B$ is more distinguishing than its stutter closure $\cong^B$.

Exercise 5.20 {} [] Prove Proposition 5.8. ■

Thus, the number of equivalence classes of $\cong^B$ may be much smaller than the number of equivalence classes of $\simeq^B$, and thus, employing $\cong^B$ for reduction may improve the efficiency of verification. Observe that the stutter-closed bisimilarity partition $\cong^B_K$ can be computed by first constructing the stutter closure K^S, and then employing the partition refinement algorithms using the propositional equivalence as the initial partition.

Example 5.11 [Stuttering equivalence of message passing protocols] Recall the modules SyncMsg of Figure 2.20 and AsyncMsg of Figure 2.24. Note that the two modules have identical observations, namely, the produced message msg_P and the consumed message msg_C. The two modules are not bisimilar. This is because the number of rounds it takes for the produced message to appear as a consumed message are different in the two modules. However, verify the two modules are stutter-closed bisimilar. ■

For the observation structure $K_{\bigcirc}$ of Figure 5.11, we have $s \cong^B t$, while the Stl formula $\exists\bigcirc p$ is satisfied only by the state t. Intuitively, the next operator allows us to count the number of rounds, while stutter-closure does not care about the number of update rounds required to change the observation. Thus, stutter-closed bisimilarity is not an abstract semantics for Stl. The next proposition asserts that Stl$^{\mathcal{U}}$ formulas cannot distinguish among $\cong^B$-equivalent states. This implies that the logic Stl$^{\mathcal{U}}$ is stutter-insensitive, and thus, for model checking of Stl$^{\mathcal{U}}$-formulas, we may use stutter-closed bisimilarity for reduction.

Proposition 5.9 [Stl$^{\mathcal{U}}$ abstraction] Stutter-closed bisimilarity $\cong^B$ is an abstract semantics for Stl$^{\mathcal{U}}$.

Exercise 5.21 {} [] (1) Prove Proposition 5.9. (2) Show that stutter-closed bisimilarity is not fully abstract for Stl$^{\mathcal{U}}$. ■

5.5 The Expressive Power of Stl

While the distinguishing powers of the logics Stl and Stl$^{\bigcirc}$ are identical, they have different expressive powers.

Let Φ and Ψ be two state logics. The logic Φ is as expressive as the logic Ψ if for every formula ϕ of Ψ, there exists a formula ψ of Φ such that for every observation structure K, the characteristic regions $[\![\phi]\!]_K$ and $[\![\psi]\!]_K$ are identical. The logic Φ is more expressive than the logic Ψ if Φ is as expressive as Ψ, but Ψ is not as expressive as Φ. The two logics Φ and Ψ are equally expressive if Φ is as expressive as Ψ, and Ψ is as expressive as Φ. The expressive powers of the two logics Φ and Ψ are incomparable if Φ is not as expressive as Ψ, and Ψ is not as expressive as Φ.

Exercise 5.22 {} [Distinguishing vs. expressive power] Let Φ and Ψ be two state logics. Prove that if Φ is as expressive as Ψ, then Φ is as distinguishing as Ψ. What other relationships between the distinguishing and expressive powers of state logics can you think of? ■

The following proposition establishes that the until-connective cannot be expressed by combining the next-connectives.

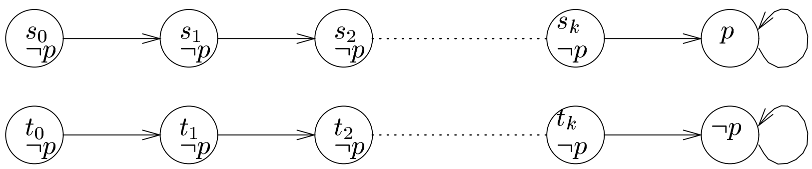

Figure 5.10: Expressive power of the eventually operator

Proposition 5.10 [Expressiveness of Until] The logic $\mathrm{Stl}^{\bigcirc}$ is not as expressive as the logic $\mathrm{Stl}^{\mathcal{U}}$.

Proof. Consider the formula $\exists\Diamond p$ of $\mathrm{Stl}^{\mathcal{U}}$. To prove that the formula $\exists\Diamond p$ is not expressible in $\mathrm{Stl}^{\bigcirc}$, we need to show that, for every formula ϕ of $\mathrm{Stl}^{\bigcirc}$, there exists an observation structure K such that $[\![\phi]\!]_K$ differs from $[\![\exists\Diamond p]\!]_K$.

Consider a formula ϕ of $\mathrm{Stl}^{\bigcirc}$, and let k be the number of occurrences of the operator $\exists\bigcirc$ in ϕ. Consider the structure shown in Figure 5.10. We have $s_0 \models \exists\Diamond p$ and $t_0 \not\models \exists\Diamond p$. We claim that for every $0 \leq i \leq k$, if ψ is a formula of $\mathrm{Stl}^{\bigcirc}$ with at most i occurrences of the temporal operator $\exists\bigcirc$, then either both s_{k-i} and t_{k-i} satisfy ψ, or both do not satisfy ψ. The proof is by induction on i.

For $i = 0$, we need to consider formulas with no occurrences of $\exists\bigcirc$, that is, propositional formulas. Since the states s_k and t_k have same observations, they agree on the truth of propositional formulas.

Now consider the case $i = j + 1$ for $0 \leq j < k$. We need to prove that the states s_{k-i} and t_{k-i} agree on the truth of formulas with at most i occurrences of $\exists\bigcirc$ assuming that the states s_{k-j} and t_{k-j} agree on the truth of formulas with at most j occurrences of $\exists\bigcirc$. The proof is straightforward using induction on the structure of ψ.

In conclusion, since ϕ has only k occurrences of $\exists\bigcirc$, the states s_0 and t_0 agree on the truth of ϕ, and hence, $[\![\phi]\!] \neq [\![\exists\Diamond p]\!]$. ■

Conversely, the next-connective cannot be expressed by combining the until-connectives.

Proposition 5.11 [Expressiveness of Next] The logic $\mathrm{Stl}^{\mathcal{U}}$ is not as expressive as the logic $\mathrm{Stl}^{\bigcirc}$.

Proof. Consider the formula $\exists\bigcirc p$ of $\mathrm{Stl}^{\bigcirc}$, and the observation structure $K_{\bigcirc}$ shown in Figure 5.11. We know that $t \models \exists\bigcirc p$ and $s \not\models \exists\bigcirc p$. We claim that the formula $\exists\bigcirc p$ is not expressible in the fragment $\mathrm{Stl}^{\mathcal{U}}$. It suffices to show that,

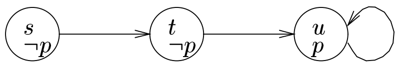

Figure 5.11: Expressive power of the next operator

for the observation structure $K_{\bigcirc}$, for every formula ϕ of $\mathrm{Stl}^{\mathcal{U}}$, either both s and t satisfy ϕ or both do not satisfy ϕ. The proof is by induction on the structure of ϕ. Since the states s and t have the same observations, they satisfy the same set of predicates. The inductive cases for logical connectives follow immediately. Let us consider the formula $\phi = \phi_1 \exists\mathcal{U} \phi_2$. From inductive hypothesis, the states s and t agree on the truth of the subformulas ϕ_1 and ϕ_2. This implies that both s and t satisfy ϕ precisely under the same conditions, namely, when they satisfy ϕ_2, or when they satisfy ϕ_1 and u satisfies ϕ_2. ∎

Exercise 5.23 {} [] Prove that there is no formula ϕ of Stl such that (1) ϕ uses only propositions, logical connectives, and the temporal operators $\exists\bigcirc$ and $\exists\Diamond$, and (2) for every observation structure K, $[\![\phi]\!]_K = [\![p\exists\mathcal{U}q]\!]_K$. This implies that the until-operator is not expressible using the next and eventually operators. ∎

Exercise 5.24 {} [Event Stl] Is Estl more expressive than Stl? Prove your answer. ∎

Exercise 5.25 {} [Strict Until] The logic STL^+ has the syntax

$$\phi \ ::= \ p \mid \phi \vee \phi \mid \neg\phi \mid \exists\bigcirc \phi \mid \phi \exists\mathcal{U}^+ \phi,$$

where the semantics of the strict-until operator is defined by the clause

$$\begin{array}{lll} s \models_K \psi\exists\mathcal{U}^+\phi & \text{iff} & \text{there is a source-}s\text{ trajectory } \overline{s}_{0..m} \text{ of } K \text{ such that (1) } m > 0 \\ & & \text{(2) } s_m \models_K \phi \text{ and} \\ & & \text{(3) for all } 0 \leq i \leq m,\ s_i \models_K \phi \vee \psi. \end{array}$$

Thus, while $p\exists\mathcal{U}q$ can be satisfied in a state s by satisfying q in s, satisfaction of $p\exists\mathcal{U}^+q$ in a state s requires a source-s trajectory that is of at least length 2 and leads to a state satisfying q. Is the logic Stl as expressive as the logic STL^+? Is the logic STL^+ as expressive as the logic Stl? ∎