Contents

Chapter 4

Graph Minimization

This chapter defines observational equivalence among states and the resulting reductions in the state-space.

4.1 Graph Partitions

State-space abstraction decreases the size of a transition graph by collapsing equivalent states. We begin by defining the quotient graphs induced by equivalence relations on the states.

> Let $G = (\Sigma, \sigma^I, \to)$ be a transition graph. An equivalence $\cong \subseteq \Sigma^2$ on the state space is called a *G-partition*. The *quotient* of G under $\cong$, denoted $G/_{\cong}$, is the transition graph $(\Sigma/_{\cong}, \sigma^I/_{\cong}, \to_{\cong})$, where $\sigma \to_{\cong} \tau$ iff there are two states $s \in \sigma$ and $t \in \tau$ such that $s \to t$.

In other words, the states of the quotient $G/_{\cong}$ are regions of the transition graph G, namely, the $\cong$-equivalence classes. A $\cong$-equivalence class is initial iff it contains an initial state. The $\cong$-equivalence class τ is a successor of the $\cong$-equivalence class σ iff a state in σ has a successor state in τ.

Let (P, p) be an invariant-verification problem. Instead of solving the reachability question $(G_P, [\![\neg p]\!])$, we choose a G_P-partition $\cong$, construct the quotient $G_P/_{\cong}$, and solve the reachability problem $(G_P/_{\cong}, [\![\neg p]\!]/_{\cong})$. If the answer to $(G_P/_{\cong}, [\![\neg p]\!]/_{\cong})$ is No, then the answer to the original question $(G_P, [\![\neg p]\!])$ is also No, and p is an invariant of the reactive module P. This verification technique is called *abstraction*, because the transition graph G_P is abstracted into the quotient $G_P/_{\cong}$ by omitting detail, such as the values of certain variables. If, on the other hand, the answer to the reachability problem $(G_P/_{\cong}, [\![\neg p]\!])$ is Yes, then p may or may not be an invariant of P. Abstraction, therefore, is a sound but incomplete verification technique for checking invariants.

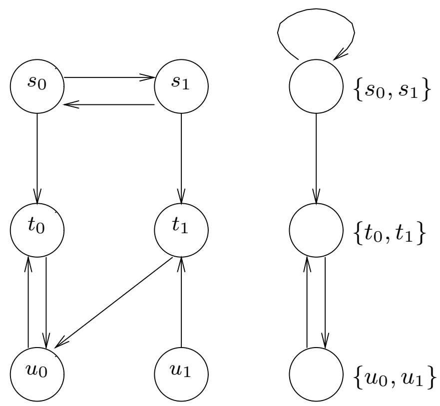

Figure 4.1: Quotient graph

Example 4.1 [Quotient graph] Consider the transition graph of Figure 4.1. The partition $\cong$ contains 3 equivalence classes $\{s_0, s_1\}$, $\{t_0, t_1\}$, and $\{u_0, u_1\}$. The corresponding quotient graph $G/_{\cong}$ has 3 states. To check whether the state s_0 is reachable from the state t_0 in G, we can check whether the state $\{s_0, s_1\}$ is reachable from the state $\{t_0, t_1\}$ in $G/_{\cong}$, and we get the correct answer No. On the other hand, to check whether the state u_1 is reachable from the state t_0 in G, we can check whether the state $\{u_0, u_1\}$ is reachable from the state $\{t_0, t_1\}$ in $G/_{\cong}$, and we get the wrong answer Yes. ■

4.1.1 Reachability-preserving Partitions

We are interested in conditions under which quotients preserve the reachability properties of a transition graph. These quotients, which are called stable, lead to abstractions that are both sound and complete for checking invariants.

The G-partition $\cong$ is *stable* if for all states s, s', and t, if $s \cong t$ and $s \to s'$, then there is a state t' such that $s' \cong t'$ and $t \to t'$. The quotient $G/_{\cong}$ is *stable* if the partition $\cong$ is stable.

In other words, for two equivalence classes σ and τ of a stable partition $\cong$,

some state in σ has a successor in τ

is equivalent to

every state in σ has a successor in τ.

Example 4.2 [Stable partition] For the transition graph of Figure 4.1, the partition $\cong$ is stable. If we add a transition to G, from state t_0 to state t_1, the partition $\cong$ will no longer be stable. ■

Suppose that $\cong$ is a stable G-partition, and let σ^T be a block of $\cong$. The reachability problem (G, σ^T) can be reduced to a reachability problem over the quotient $G/_{\cong}$, whose state space may be much smaller than the state space of G. Indeed, $\Sigma/_{\cong}$ may be finite for infinite Σ.

Theorem 4.1 [Stable partitioning] *Let $G/_{\cong}$ be a stable quotient of the transition graph G, and let σ be a block of $\cong$. Then the two reachability problems (G, σ) and $(G/_{\cong}, \sigma/_{\cong})$ have the same answer.*

Proof. If the answer to (G, σ^T) is YES, then the answer to $(G/_{\cong}, \sigma^T/_{\cong})$ is also YES. This direction does not require $\cong$ to be stable or σ^T to be a block of $\cong$.

Suppose the answer to $(G/_{\cong}, \sigma^T/_{\cong})$ is YES. Consider the witness trajectory $\sigma_0 \rightarrow_{\cong} \cdots \rightarrow_{\cong} \sigma_m$ in $G/_{\cong}$ with $\sigma_o \in \sigma^I/_{\cong}$ and $\sigma_m \in \sigma^T/_{\cong}$. Since σ^T is a block of $\cong$, we know that $\sigma_m \subseteq \sigma^T$. Choose a state s_0 in the intersection $\sigma^I \cap \sigma_0$. Since $\cong$ is stable, we know that whenever $\tau \rightarrow_{\cong} \upsilon$, for all state $s \in \tau$, there exists a state $t \in \upsilon$ such that $s \rightarrow t$. Starting with $s_0 \in \sigma_0$, choose states $s_1, \ldots s_m$, one by one, such that, for every $1 \leq i \leq m$, $s_i \in \sigma_i$, and $s_{i-1} \rightarrow s_i$. Since s_m is in the target region σ^T, the trajectory $\overline{s}_{0 \ldots m}$ is a witness to the reachability question (G, σ^T). ■

Exercise 4.1 {`T2`} [Inverse stability] The G-partition $\cong$ is *initialized* if the initial region σ^I is a block of $\cong$. The G-partition $\cong$ is *backstable* if for all states s, s', and t, if $s \cong t$ and $s' \rightarrow s$, then there is a state t' such that $s' \cong t'$ and $t' \rightarrow t$. Equivalently, $\cong$ is a backstable G-partition iff $\cong$ is a stable G^{-1}-partition.

Let G be a transition graph, let $\cong$ be an initialized backstable G-partition, and let σ^T be a region of G. Prove that the two reachability problems (G, σ^T) and $(G/_{\cong}, \sigma^T/_{\cong})$ have the same answer. ■

Projecting states of a module to a subset of variables gives a partition of the underlying transition graph. Let P be a module, and let X be a subset of its variables. For two states s and t of P, let $s \cong_{[X]} t$ if $X[s] = X[t]$. The equivalence $\cong_{[X]}$ is a G_P-partition.

Example 4.3 [Latched variables] Recall the definition of latched variables $latchX_P$ of a module P. The G_P-partition $\cong_{[latchX_P]}$ is a stable partition, and the reduced transition graph G_P^L is the resulting quotient graph. ■

Thus, projection to latched variables results in a stable partition. An orthogonal method to obtain a stable partition is to find a set of variables that is closed under dependencies.

> STABLE VARIABLE SETS
>
> Let P be a module. A subset $X \subseteq X_P$ of the module variables is *stable* if for every variable $x \in X$, if the variable x is controlled by the atom U of P then both $\mathsf{read}X_U \subseteq X$ and $\mathsf{await}X_U \subseteq X$.

Proposition 4.1 [Stable projections] *Let P be a module, and let X be a stable subset of its variables. Then, the G_P-partition $\cong_{[X]}$ is a stable partition.*

Exercise 4.2 {T2} [Elimination of redundant variables] Consider the invariant verification problem (P, p). Let X be a stable set of the module variables. Show that if the observation predicate p refers only to the variables in $X \cap \mathsf{obs}X_P$, then $[\![\neg p]\!]$ is a block of the G_P-partition $\cong_{[X]}$. Then, the invariant verification problem (P, p) reduces to the reachability problem $(G_P/_{\cong_{[X]}}, [\![\neg p]\!])$.

Give an algorithm to compute the minimal set X_p of variables that contains all the variables in p and is stable. Notice that the set X_p contains all the variables whose initialization and update influences the initialization and update of the variables in p, and thus, the remaining variables $X_P \setminus X_p$ are redundant for the verification of p. ■

4.1.2 Graph Symmetries

Stable quotients often arise from exploiting the symmetries of a transition graph. For instance, in the module *Pete* the individual processes are symmetric, resulting in the symmetry in the state-space of G_{Pete}. To formalize the reduction afforeded by symmetries, we beign by defining graph automorphisms. A graph automorphism is a one-to-one onto mapping from vertices to vertices that preserves the initial region as well as the transitions.

> GRAPH AUTOMORPHISM
>
> Consider two transition graphs $G_1 = (\Sigma_1, \sigma_1^I, \rightarrow_1)$ and $G_2 = (\Sigma_2, \sigma_2^I, \rightarrow_2)$. A bijection f from Σ_1 to Σ_2 is an *isomorphism* from G_1 to G_2 if (1) $f(\sigma_1^I) = \sigma_2^I$ and (2) for all states $s, t \in \Sigma_1$, $s \rightarrow_1 t$ iff $f(s) \rightarrow_2 f(t)$. An isomorphism from G to G is called a *G-automorphism.*

Remark 4.1 [Graph Automorphisms] For every transition graph G, the identity function is a G-automorphism. If f is a G-automorphism, then so is its inverse f^{-1}. The (functional) composition of two G-automorphisms is a G-automorphism. The set of all binary functions over the state-space of G forms a group under functional composition. The set of G-automorphisms forms a subgroup. Furthermore, for a set F of G-automorphisms, the subgroup generated by F contains only G-automorphisms. ■

The group of G-automorphisms under functional composition is called the *symmetry group* of G. This symmetry group, or any of its subgroups, can be used to define a stable partition. Given a set F of generators that are G-automorphisms, we obtain the corresponding symmetric partition by considering all the automorphisms in the subgroup generated by F.

> SYMMETRIC PARTITION
>
> Let G be a transition graph, and let F be a set of G-automorphisms. The F*-symmetric partition* $\cong^F$ is defined by: for all states s and t of G, let $s \cong^F t$ if there is an automorphism $f \in \mathit{closure}(F)$ such that $t = f(s)$.

Stability of the symmetric partition follows immediately from the definitions.

Theorem 4.2 [Symmetric partitioning] *Let G be a transition graph, and let F be a set of G-automorphisms. The induced G-partition $\cong^F$ is stable.*

Let G be a transition graph, let σ be a region of G, and let F be a set of G-automorphisms. If σ is a block of the induced G-partition $\cong^F$, then the quotient $G/_{\cong^F}$ can be used to solve the reachability problem (G, σ). Notice that σ is a block of $\cong^F$ iff for all G-automorphisms $f \in F$, $f(\sigma) = \sigma$.

Example 4.4 [Symmetry of mutual exclusion] Recall Peterson's mutual-exclusion protocol from Chapter 1. Consider the following bijection f on the state space Σ_{Pete} of the underlying transition graph G_{Pete}: let $t = f(s)$ iff

$x_1[t] = x_2[s]$ and $x_2[t] \neq x_1[s]$, and
$pc_1[t] = pc_2[s]$ and $pc_2[t] = pc_1[s]$.

The function f swaps the values of pc_1 and pc_2, swaps the values of x_1 and x_2, and toggles x_2. Note that the thruth of the condition $x_1 = x_2$ is toggled by the function f.

Verify that the function f is a G_{Pete}-automorphism. The composition $f \circ f$ simply toggles both x_1 and x_2: $t = f \circ f(s)$ iff

$x_1[t] \neq x_1[s]$ and $x_2[t] \neq x_2[s]$, and
$pc_1[t] = pc_1[s]$ and $pc_2[t] = pc_2[s]$.

The composition $f \circ f \circ f$ is the inverse of f: $t = f \circ f \circ f(s)$ iff

$x_1[t] \neq x_2[s]$ and $x_2[t] = x_1[s]$, and
$pc_1[t] = pc_2[s]$ and $pc_2[t] = pc_1[s]$.

It follows that f^4 equals the identity map. Consequently, the subgroup $closure(f)$ generated by the automorphism f equals $\{f, f \circ f, id, f^{-1}\}$, where id the identity function. Consider the four initial states—s_1, s_2, s_3, and s_4—of *Pete*

$$\begin{array}{l} s_1(pc_1) = outC,\ s_1(x_1) = true,\ s_1(pc_2) = outC,\ s_1(x_2) = true; \\ s_2(pc_1) = outC,\ s_2(x_1) = true,\ s_2(pc_2) = outC,\ s_2(x_2) = false; \\ s_3(pc_1) = outC,\ s_3(x_1) = false,\ s_3(pc_2) = outC,\ s_3(x_2) = true; \\ s_4(pc_1) = outC,\ s_4(x_1) = false,\ s_4(pc_2) = outC,\ s_4(x_2) = false. \end{array}$$

Verify that $s_2 = f(s_1)$, $s_3 = f(s_2)$, $s_4 = f(s_3)$, and $s_1 = f(s_4)$. In the partition $\cong^f$, two states are equivalent if one can be obtained from the other by applying one of the automorphisms in $closure(f)$. In particular, all the four initial states are equivalent. Verify that while G_{Pete} contains 36 states, the partition $\cong^f$ contains 9 classes; while the reachable subgraph of G_{Pete} contains 20 states, the number of reachable classes of $\cong^F$ is 5.

The region $[\![\neg(pc_1 = inC \wedge pc_2 = inC)]\!]$ is invariant under the function f, and hence, is a block of the stable partition $\cong^f$. It follows that the quotient $G_{Pete}/_{\cong^f}$ can be used to check that the protocol *Pete* enforces mutual exclusion. ■

In practice, the communication topology among different components yields graph automorphisms. Two typical examples are:

- *Star Topology:* The system consists of a module P (server) communicating with modules $P_1, \ldots P_n$ (clients). The client modules $P_1, \ldots P_n$ are renamed copies of each other, and thus, there is a one-to-one correspondence between the controlled variables of two client modules. Two client modules do not have any common variables, and thus, each client module communicates only with the server. In this situation, swapping the values of the controlled variables of two client modules results in an automorphism. In particular, the set F of generators contains for every pair $1 \leq i, j \leq n$, the automorphism f_{ij} that swaps the values of the controlled variables of P_i with the values of the corresponding controlled variables of P_j.

- *Ring Topology:* The system consists of modules $P_1, \ldots P_n$ connected in a ring, that is, every module P_i communicates only with its neighboring modules P_{i-1} and P_{i+1} (where increments and decrements are modulo n). All the modules are renamed copies of each other. In this situation, every rotation of the indices yields an automorphism. That is, for every i, the function f_i is an automorphism, where $t = f_i[s]$ if the values of the controlled variables of the module P_j in state t equal the values of the corresponding controlled variables of the module P_{j+i} in state s.

Exercise 4.3 {T2} [Symmetry in Railroad controller] Consider the module *RailroadSystem* from Chapter 2. Find a suitable set F of automorphisms. What is the equivalence $\cong^F$ induced on the state-space? ■

To apply symmetric reduction to the invariant verification problem (P, p), we first find a suitable set of G_P-automorphisms. The next step is to find a mapping *rep* that maps every state s to a unique representative of the equivalence class of $\cong^F$ that contains s: if $s \cong^F t$, then $rep(s) = rep(t)$. If we have such a function *rep*, then the depth-first search algorithm is modified so that only the representative states are explored. This is achieved by replacing the initial region σ^I by the set $rep(\sigma^I)$ of representative initial states, and replacing the successor function *post* by the function $rep \circ post$ that considers only representative states. Consequently, the complexity of the search is proportional the size of the quotient graph with respect to $\cong^F$.

Exercise 4.4 {T3} [Representative states in mutual exclusion] Consider the automorphism f, and the induced equivalence $\cong^f$, on the state-space of the module *Pete* considered in Example 4.4. Suggest a suitable set of representative states and the function *rep* that maps each state to its representative. ■

4.2 Partition Refinement

Suppose we wish to solve multiple verification problems involving a transition graph G. Then, it is prudent to find a stable G-partition $\cong$ such that there are as few $\cong$-equivalence classes as possible. Then the quotient $G/_{\cong}$ can be used to solve the verification problems concerning G.

4.2.1 The Structure of Stable Partitions

If $\cong_1$ and $\cong_2$ are stable partitions of a transition graph, then so is their join:

Lemma 4.1 [Union-closure of stable partitions] *Let G be a transition graph. If E is a set of stable G-partitions, then the join $\bigcup^* E$ is a stable G-partition.*

Proof. Let G be a transition graph, let E is a set of stable G-partitions, and let $\cong$ be the join $\bigcup^* E$ of all partitions in E. Suppose $s \cong t$ and $s \rightarrow s'$. Since $\cong$ is the transitive closure of the union of the equivalence relations in E, there are states $s_0, \ldots s_n$ and partitions $\cong_1, \cong_n$ in E such that $s_0 = s$, $s_n = t$, and $s_{i-1} \cong_i s_i$ for $1 \leq i \leq n$. Let $s'_0 = s'$. We have $s_0 \rightarrow s'_0$. Since each partition $\cong_i$ is stable, by induction on i, there exist states $s'_1, \ldots s'_n$ such that for $1 \leq i \leq n$, $s_i \rightarrow s'_i$ and $s'_{i-1} \cong_i s'_i$. Choose $t' = s'_n$. We have $t \rightarrow t'$ and $s' \cong t'$. ■

Corollary 4.1 [CPO of stable partitions] *For every transition graph G, the refinement relation $\preceq$ is a complete lattice on the stable G-partitions.*

Exercise 4.5 {T2} [Complete lattice of stable partitions] Consider the complete lattice $\preceq$ on the stable G-partitions of the transition graph G. Let E be a set of stable G-partitions. The least upper $\preceq$-bound for E is the join $\bigcup^* E$. What is the greatest lower $\preceq$-bound for E? ■

Let $\cong$ be a partition of the transition graph G. Consider the set E of all stable partitions that refine $\cong$. The join $\bigcup^* E$, which is guaranteed to exist, is a stable partition. Furthermore, since every partition in E refines $\cong$, so does $\bigcup^* E$. Consequently, the join $\bigcup^* E$ is the coarsest partition that is both stable and is finer than $\cong$.

> Let G be a transition graph, and let $\cong$ be a G-partition. The *coarsest stable refinement* of $\cong$, denoted $\min_G(\cong)$, is the join of all stable G-partitions that refine $\cong$. The quotient $G/_{\min(\cong)}$ is called $\cong$-*minimal*.

It follows that $\min(\cong)$ is a stable G-partition that refines $\cong$, and that all stable G-partitions that refine $\cong$ also refine $\min(\cong)$.

Remark 4.2 [Refinement of identity and universal partitions] If $\cong$ is the identity partition (i.e. all equivalence classes of $\cong$ are singletons), then $\min(\cong)$ equals $\cong$. If G is a serial transition graph, and $\cong$ is the universal partition (i.e. contains a single equivalence class containing all states), then $\min(\cong)$ equals $\cong$. ■

The partition-refinement problem

> An instance $(G, \cong^I)$ of the *partition-refinement problem* consists of (1) a transition graph G and (2) [the *initial partition*] a G-partition $\cong^I$. The answer to the partition-refinement problem $(G, \cong^I)$ is the coarsest stable refinement $\min(\cong^I)$ of the initial partition $\cong^I$.

Example 4.5 [Coarsest stable refinement] Consider the transition graph of Figure 4.1. Suppose the initial partition $\cong^I$ contains two regions; $\{s_0, s_1, t_0, t_1\}$ and $\{u_0, u_1\}$. The initial partition itself is not stable. Its coarsest stable refinement contains three regions $\{s_0, s_1\}$, $\{t_0, t_1\}$, and $\{u_0, u_1\}$. ■

Minimal reachability-preserving quotients

Let G be a transition graph with the state space Σ. For a region σ of G, let $\cong^\sigma$ denote the binary G-partition $\{\sigma, \Sigma \backslash \sigma\}$. The partition $\cong^\sigma$ is the coarsest partition that has σ as a block. The $\cong^\sigma$-minimal quotient $G/_{\min(\cong^\sigma)}$ of the transition graph G can be used to solve the reachability problem (G, σ), because $\min(\cong^\sigma)$ is stable and σ is a block of $\min(\cong^\sigma)$. For a set R of regions, let $\cong^R$ denote the G-partition $(\cap \sigma \in R.\ \cong^\sigma)$. The $\cong^R$-minimal quotient $G/_{\min(\cong^R)}$

can then be used to solve all reachability problems of the form (G, σ) for $\sigma \in R$. For example, let P be a module. Copnsider the equivalence $\cong$ on the state-space of P induced by the observations: $s \cong t$ iff $\mathsf{obs}X_P[s] = \mathsf{obs}X_P[t]$. Then, two $\cong$-equivalent states satisfy the same set of observation predicates. The quotient $G_P/_{\min(\cong)}$ can then be used to solve all invariant-verification problems for the module P.

Exercise 4.6 {`T3`} [Reachable portion of minimal quotients] Let G be a transition graph, and let σ be a region of G. To solve the reachability problem (G, σ), it suffices to consider the reachable region σ^R of G. We may first find a minimal quotient of G and then construct the reachable subquotient, or we may first construct the reachable subgraph of G and then find a minimal quotient. Both methods lead to isomorphic results. Let G_1 be the reachable subgraph of the $\cong^\sigma$-minimal quotient $G/_{\min(\cong^\sigma)}$ of G. For $\cong = \{\sigma^R \cap \sigma, \sigma^R \backslash \sigma\}$, let G_2 be the $\cong$-minimal quotient $G^R/_{\min(\cong)}$ of the reachable subgraph G^R of G. Prove that the two transition graphs G_1 and G_2 are isomorphic. ■

Exercise 4.7 {`T4`} [Inverse minimal quotients] Let $G = (\Sigma, \sigma^I, \rightarrow)$ be a transition graph, and let $\cong$ be a G-partition. The *coarsest backstable refinement* of $\cong$, denoted $\min^{-1}(\cong)$, is the join of all backstable G-partitions that refine $\cong$. The quotient $G/_{\min^{-1}(\cong^{\sigma^I})}$ of the transition graph G can be used to solve the reachability problem (G, σ), for any region σ of G. Prove that the unreachable region $\Sigma \backslash \sigma^R$ is a $\min^{-1}(\cong^{\sigma^I})$-equivalence class.

Let σ be a region of G. To solve the reachability problem (G, σ), we may compute (the reachable portion of) a stable refinement of $\cong^\sigma$, or a backstable refinement of $\cong^{\sigma^I}$. Depending on the given reachability problem, either method may be superior to the other method. Consider two state spaces: (A) the quotient $\sigma^R/_{\min(\cong^\sigma)}$ of the reachable region σ^R with respect the coarsest stable refinement $\min(\cong^\sigma)$; (B) the coarsest backstable refinement $\min^{-1}(\cong^{\sigma^I})$. Give an example of a reachability problem for which state space (A) is finite and state space (B) is infinite, and an example for which state space (A) is infinite and state space (B) is finite. ■

Exercise 4.8 {`T3`} [Symbolic reachability versus partition refinement] Let G be a transition graph, and let σ be a region of G. Prove that for all natural numbers i, the region $\mathit{pre}^i(\sigma)$ is a block of the coarsest stable refinement $\min(\cong^\sigma)$, and the region $\mathit{post}^i(\sigma^I)$ is a block of the coarsest backstable refinement $\min^{-1}(\cong^{\sigma^I})$. Conclude that if the coarsest backstable refinement $\min^{-1}(\cong^{\sigma^I})$ is finite, then the transition graph G is finitely reaching. ■

4.2.2 Partition-refinement Algorithms

We first develop a schematic algorithm for solving the partition-refinement problem, and prove it correct. For a running-time analysis, we then present several concrete instantiations of the schematic partition-refinement algorithm.

A region characterization of stability

Let G be a transition graph, and let σ and τ be two regions of G. The region σ is *stable with respect to* the region τ if either $\sigma \subseteq pre(\tau)$ or $\sigma \cap pre(\tau) = \emptyset$. Let $\cong$ be a G-partition. The partition $\cong$ is *stable with respect to* the region τ if all $\cong$-equivalence classes are stable with respect to τ. This region-based definition gives an alternative characterization of stability.

Lemma 4.2 [Stability with respect to regions] *Let G be a transition graph, and let $\cong$ be a G-partition. Then $\cong$ is stable iff $\cong$ is stable with respect to all $\cong$-equivalence classes.*

Stabilization of a partition with respect to a region

Partition-refinement algorithms stabilize the given initial partition by repeatedly splitting equivalence classes. Consider a partition $\cong$. If $\cong$ is not stable, then, by Lemma 4.2, there are two equivalence classes σ and τ of $\cong$ such that σ is not stable with respect to τ. In such a case, we can split σ into two regions, one that contains states which have successors in τ and the other one that contains states with no successors in τ. That is, we split σ at the boundary of the predecessor region of τ.

Let τ be a region of the transition graph G. For a region σ of G, let

$$Split(\sigma, \tau) \;=\; \begin{cases} \{\sigma\} & \text{if } \sigma \subseteq \tau \text{ or } \sigma \cap \tau = \emptyset, \\ \{\sigma \cap \tau, \sigma \backslash \tau\} & \text{else,} \end{cases}$$

be the result of splitting σ at the boundary of τ. For a G-partition $\cong$, let

$$Split(\cong, \tau) \;=\; (\cup \sigma \in \cong . \, Split(\sigma, \tau))$$

be the result of splitting $\cong$ at the boundary of τ. The result $Split(\cong, \tau)$ is a G-partition that refines $\cong$ and contains at most twice as many equivalence classes as $\cong$. To stabilize $\cong$ with respect to τ, we split $\cong$ at the boundary of $pre(\tau)$:

$$Stabilize(\cong, \tau) \;=\; Split(\cong, pre(\tau)).$$

The stablization of a region σ with respect to τ is depicted pictorially in Figure 4.2. The *Stabilize* operation can be implemented either symbolically or enumeratively.

Symbolic stabilization. Suppose that the region τ is given by a symbolic region representation $\{\tau\}_s$, and the partition $\cong$ is given by a list $\langle \{\sigma\}_s \mid \sigma \in \cong \rangle$

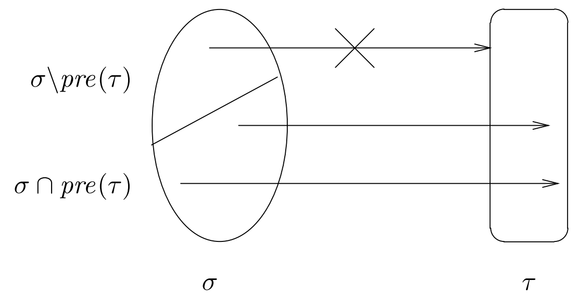

Figure 4.2: Stabilizing one region with respect to another

of symbolic region representations. The operation *Stabilize*$(\cong, \tau)$ can then be performed using boolean operations, emptiness checking, and the *pre* operation on symbolic region representations.

Enumerative stabilization. We are given an abstract data type **partition** that maintains a collection of nonempty, disjoint subsets of the state-space. The data type **partition** is like **set of region**, but supports the following additional operations:

Find: For a state s and a partition $\cong$, the operation *Find*$(s, \cong)$ returns the (name of) the region that contains s, if such a region exists; otherwise *Find*$(s, \cong)$ returns *nil*.

Create: For a state s and a partition $\cong$, the operation *Create*$(s, \cong)$ removes s from any existing region in $\cong$, creates a singleton set containing s, and returns the newly created set. Note that *Create*$(s, \cong)$ destructively updates the partition $\cong$.

Move: For a state s, a partition $\cong$, and a set σ in $\cong$, the operation *Move*$(s, \sigma, \cong)$ removes s from any existing set in $\cong$ and adds s to the set σ; if the result of removing s from an existing set results in an empty set, that set is destroyed.

Exercise 4.9 {P2} [Abstract data type **partition**] Implement the abstract data type **partition** so that each of the three operations *Find*, *Create*, and *Move* take constant time. ■

Let G be a finite transition graph. Suppose that the region τ is given by a list $\{\tau\}_e$ of states, and the partition $\cong$ is given using the abstract data type **partition**. Furthermore, with each state s we are given a list of all predecessor states in *pre*(s), and with each set σ in **partition** we are given the name *new*(σ) of another set in **partition**. When stabilizing the region σ with respect to τ, the states in $\sigma \cap pre(\tau)$ are moved to the set $new(\sigma)$. Initially, all *new* pointers are *nil*, and they are reset after stabilization. The operation $Stabilize(\cong, \tau)$ can then be performed as follows:

> **foreach** $s \in \tau$ **do foreach** $t \in pre(s)$ **do** $Update(t, \cong)$ **od od**;
> **foreach** $s \in \tau$ **do foreach** $t \in pre(s)$ **do** $Reset(t, \cong)$ **od od**,

where both

> $Update(t, \cong)$:
> **if** $new(Find(t, \cong)) = nil$
> **then** $new(Find(t, \cong)) := Create(t, \cong)$
> **else** $Move(t, new(Find(t, \cong)), \cong)$
> **fi**

and

> $Reset(t, \cong)$:
> $new(Find(t, \cong)) := nil$

take constant time. Let n_τ be the number of states in the region τ, and let m_τ be the number of transitions whose target lies in τ. The time required by the operation $Stabilize(\cong, \tau)$ is $stabcost(\tau) = O(m_\tau + n_\tau)$. We charge the stabilization cost $stabcost(\tau)$ to the individual states in τ. If m_s is the number of transitions with target s (i.e. $m_s = |pre(s)|$), then we charge $stabcost(s) = O(m_s + 1)$ to each state $s \in \tau$. Then $stabcost(\tau) = (+s \in \tau.\, stabcost(s))$.

Iterative stabilization of a partition

The key properties of the operation of stabilizing a partition with respect to a region are summarized in the next lemma.

Lemma 4.3 [Stabilization for partition refinement] *Let G be a transition graph, let $\cong$ be a G-partition, and let τ be a region of G. (1) If τ is a block of $\cong$, then* $\min(\cong) \preceq Stabilize(\cong, \tau)$. *(2) Every G-partition that refines $Stabilize(\cong, \tau)$ is stable with respect to τ.*

Exercise 4.10 {`T2`} [Stabilization for partition refinement] Prove Lemma 4.3.
■

Lemma 4.3 suggests a partition-refinement algorithm that, starting from the given initial partition, repeatedly stabilizes the partition with respect to one of its blocks. Part (1) ensures that stabilization with respect to a block never causes unnecessary splitting. Part (2) ensures that every block needs to be considered for stabilization at most once. The resulting scheme is shown in Figure 4.3.

In Algorithm 4.1, at the beginning of each execution of the while-loop, we know that

Algorithm 4.1 [Schematic Partition Refinement]

Input: a transition graph $G = (\Sigma, \sigma^I, \rightarrow)$ and an initial G-partition $\cong^I$.
Output: the coarsest stable refinement $\min(\cong^I)$.
Local: a G-partition $\cong$ and a region set *done*.

$\cong := \cong^I$; *done* $:= \{\Sigma\}$;
while $\cong \not\subseteq$ *done* **do**
  $\{$**assert** $\min(\cong^I)$ refines $\cong$, and $\cong$ is stable w.r.t. all regions in *done*$\}$
  Choose a block τ of $\cong$ such that $\tau \notin$ *done*;
  $\cong :=$ *Stabilize*$(\cong, \tau)$;
  done $:=$ *Insert*$(\tau$, *done*$)$
  od;
return $\cong$.

Figure 4.3: Partition refinement

1. the coarsest stable refinement $\min(\cong^I)$ is a refinement of the current partition $\cong$,
2. every region in the set *done* is a block of the current partition $\cong$, and
3. the current partition $\cong$ is stable with respect to every region in *done*.

Algorithm 4.1 terminates iff $\min(\cong^I)$ has finitely many equivalence classes. Suppose that $\min(\cong^I)$ has n equivalence classes and, therefore, 2^n blocks. With every iteration of the while-loop, a block of $\min(\cong^I)$ is added to the set *done*. It follows that the while-loop is executed at most 2^n times.

Theorem 4.3 [Schematic partition refinement] *Let G be a transition graph, and let $\cong^I$ be a G-partition. If the coarsest stable refinement* $\min(\cong^I)$ *is finite, then Algorithm 4.1 solves the partition-refinement problem* $(G, \cong^I)$.

A quadratic partition-refinement algorithm

If we carefully choose the region τ in each iteration of Algorithm 4.1, we obtain polynomial-time implementations. A quadratic running time is achieved if, during consecutive iterations, we systematically stabilize the initial partition $\cong^I$ first with respect to all $\cong^I$-equivalence classes, then with respect to all equivalence classes of the resulting partition, etc. The resulting algorithm is shown in Figure 4.4.

Observe that during the execution of the for-loop, every equivalence-class of $\cong^{prev}$ is a block of the current partition $\cong$, and at the beginning of the while-loop the current partition $\cong$ is stable with respect to every region in $\cong^{prev}$. Thus,

Algorithm 4.2 [Quadratic Partition Refinement]

Input: a transition graph $G = (\Sigma, \sigma^I, \rightarrow)$ and an initial G-partition $\cong^I$.
Output: the coarsest stable refinement $\min(\cong^I)$.
Local: two G-partitions $\cong$ and $\cong^{prev}$.

$\cong := \cong^I$; $\cong^{prev} := \{\Sigma\}$;
while $\cong \neq \cong^{prev}$ **do**
  {**assert** $\min(\cong^I)$ refines $\cong$, and $\cong$ is stable w.r.t. all regions in $\cong^{prev}$}
  $\cong^{prev} := \cong$;
  for each $\tau \in \cong^{prev}$ **do** $\cong := \mathit{Stabilize}(\cong, \tau)$ **od**
  od;
return $\cong$.

Figure 4.4: Quadratic algorithm for partition refinement

Algorithm 4.2 is an instance of Algorithm 4.1, and its correctness follows immediately. With every iteration of the while-loop, the number of $\cong$-equivalence classes increases. Hence, if $\min(\cong^I)$ has n equivalence classes, the while-loop is executed at most n times. The for-loop can be implemented either symbolically or enumeratively. Consider an enumerative implementation of Algorithm 4.2 for an input graph G with n states and $m \geq n$ transitions. Then the coarsest stable refinement $\min(\cong^I)$ has at most n equivalence classes, and the time required by the for-loop is

$$\begin{aligned}(+\tau \in \cong^{prev} .\, \mathit{stabcost}(\tau)) &= (+s \in \Sigma.\, \mathit{stabcost}(s)) \\ &= (+s \in \Sigma.\, O(m_s + 1)) \\ &= O(m).\end{aligned}$$

Theorem 4.4 [Quadratic partition refinement] *Let G be a finite transition graph with n states and m transitions. The running time of Algorithm 4.2 on input G is $O(m \cdot n)$.*

Exercise 4.11 {`P3`} [Quadratic partition refinement] Write a program that implements Algorithm 4.2 symbolically, and a program that implements Algorithm 4.2 enumeratively. For your symbolic program, assume that the input graph G is given by a symbolic graph representation $\{G\}_s$, and the input partition $\cong^I$ is given by a list $\langle \{\sigma\}_s \mid \sigma \in \cong^I \rangle$ of symbolic region representations. For your enumerative program, assume that the input graph G is given by an enumerative graph representation $\{G\}_e$, and the input partition $\cong^I$ is given by a list $\langle \{\sigma\}_e \mid \sigma \in \cong^I \rangle$ of enumerative region representations. The asymptotic running time of your enumerative program should be quadratic in the size of the input. ∎

Algorithm 4.3 [Paige-Tarjan Partition Refinement]

Input: a transition graph $G = (\Sigma, \sigma^I, \rightarrow)$, and an initial G-partition $\cong^I$.
Output: the coarsest stable refinement $\min(\cong^I)$.
Local: two G-partitions $\cong$ and $\cong^{done}$.

$\cong := \cong^I$; $\cong^{done} := \{\Sigma\}$;
while $\cong \subset \cong^{done}$ **do**
  {**assert** $\min(\cong^I) \preceq \cong$, and $\cong$ is stable w.r.t. all regions in $\cong^{done}$}
  Choose $\sigma \in (\cong^{done} \setminus \cong)$;
  Choose $\tau \in \cong$ such that $\tau \subseteq \sigma$ and $|\tau| \leq |\sigma|/2$;
  $\cong := \mathit{Stabilize}(\mathit{Stabilize}(\cong, \tau), \sigma \backslash \tau)$;
  $\cong^{done} := \mathit{Insert}(\sigma \backslash \tau, \mathit{Insert}(\tau, \mathit{Delete}(\sigma, \cong^{done})))$
  od;
return $\cong$.

Figure 4.5: Paige-Tarjan algorithm for partition refinement

The Paige-Tarjan partition-refinement algorithm

To improve the time complexity of partition refinement, we need an improved strategy to choose the splitting block. The number of stabilization operations required can equal the number of equivalence-classes in the coarsest stable refinement, which can, in turn, be equal to the number of states in the transition graph, in the worst case.

Exercise 4.12 {T3} [Worst-case for quadratic partition refinement] Give an instance $(G, \cong^I)$ of the partition refinement problem such that the execution of Algorithm 4.1 on this instance, requires n iterations of the while-loop, irrespective of the choices of the splitting blocks τ. ■

If, at each iteration of Algorithm 4.1, we carefully choose a "small" block τ of $\cong$ for the operation $\mathit{Stabilize}(\cong, \tau)$, we arrive at a subquadratic running time. A suitable criterion for "small" is that τ is a $\cong$-equivalence class that contains at most half the states of any $\cong$-block σ if $\cong$ is known to be stable with respect to σ. This criterion is enforced by maintaining a second partition, $\cong^{done}$, such that $\cong$ refines $\cong^{done}$ and is stable with respect to all $\cong^{done}$-equivalence classes. The algorithm is shown in Figure 4.5. Observe that Algorithm 4.3 is an instance of Algorithm 4.1.

Consider an enumerative implementation of Algorithm 4.3 for an input graph G with n states and $m \geq n$ transitions. Since the number of $\cong^{done}$-equivalence classes increases with every iteration, the while-loop is executed at most n times. Let σ_i and τ_i denote the equivalence classes of $\cong^{done}$ and $\cong$, respectively, that are chosen in the i-th iteration of the while-loop. An appropriate choice of τ_i can be performed by maintaining for each $\cong$-equivalence class υ a counter that indicates the number of states in υ. Suppose that a state $s \in \Sigma$ belongs to both τ_i and τ_j, for $j > i$. Since $\sigma_j \subseteq \tau_i$ and $|\tau_j| \leq |\sigma_j|/2$, also $|\tau_j| \leq |\tau_i|/2$. It follows that there are at most $\log n + 1$ iterations i such that $s \in \tau_i$.

The i-th iteration of the while-loop consists of two stabilizing operations, one with respect to τ_i and one with respect to $\sigma_i \backslash \tau_i$. Since each state belongs only to $O(\log n)$ many regions τ_i, the cumulative cost of the stabilization operations with respect to all regions τ_i is $(+s \in \Sigma.\, O(\log n) \cdot \mathit{stabcost}(s)) = O(m \cdot \log n)$. A state, however, may belong to $O(n)$ many regions of the form $\sigma_i \backslash \tau_i$. The following lemma states that to stabilize $\cong$ with respect to $\sigma_i \backslash \tau_i$, instead of splitting $\cong$ with respect to $\mathit{pre}(\sigma_i \backslash \tau_i)$, we can split it with respect to $\mathit{pre}(\tau_i) \backslash \mathit{pre}(\sigma_i \backslash \tau_i)$, thereby, avoiding the computation of $\mathit{pre}(\sigma_i \backslash \tau_i)$. This observation allows us to implement the operation $\mathit{Stabilize}(\cong, \sigma_i \backslash \tau_i)$ in time $\mathit{stabcost}(\tau_i)$, that is, at the same cost as the operation $\mathit{Stabilize}(\cong, \tau_i)$.

Lemma 4.4 [Efficient stabilization for Paige-Tarjan] *Let G be a transition graph, let $\cong$ be a G-partition, and let σ and τ be two blocks of $\cong$. If $\cong$ is stable with respect to σ and τ, then*

$$\mathit{Stabilize}(\cong, \sigma \backslash \tau) \;=\; \mathit{Split}(\cong, \mathit{pre}(\tau) \backslash \mathit{pre}(\sigma \backslash \tau)).$$

Exercise 4.13 {T3} [Efficient stabilization for Paige-Tarjan] Prove Lemma 4.4. ■

For every state $s \in \Sigma$ and every $\cong^{done}$-equivalence class σ, we maintain a counter $\mathit{tcount}(s, \sigma)$ that indicates the number of transitions from s to a state in σ; that is, $\mathit{tcount}(s, \sigma) = |\sigma \cap \mathit{post}(s)|$. The operation $\mathit{Stabilize}(\cong, \sigma \backslash \tau)$ can then be performed in time $\mathit{stabcost}(\tau) = O(m_\tau + n_\tau)$:

```
foreach s ∈ τ do
  foreach t ∈ pre(s) do tcount(t, τ) := 0 od
  od;
foreach s ∈ τ do
  foreach t ∈ pre(s) do tcount(t, τ) := tcount(t, τ) + 1 od
  od;
foreach s ∈ τ do
  foreach t ∈ pre(s) do
    tcount(t, σ\τ) := tcount(t, σ) − tcount(t, τ);
    if tcount(t, σ\τ) = 0 then Update(t, ≅) fi
    od
  od;
foreach s ∈ τ do foreach t ∈ pre(s) do Reset(t, ≅) od od.
```

If we charge the cost of both parts of the operation $\mathit{Stabilize}(\mathit{Stabilize}(\cong, \tau), \sigma\backslash\tau)$ to the states in τ, it follows that the time required by Algorithm 4.3 is

$$(+s \in \Sigma.\, O(\log n) \cdot 2 \cdot \mathit{stabcost}(s)) \;=\; O(m \cdot \log n).$$

Theorem 4.5 [Paige-Tarjan partition refinement] *Let G be a finite transition graph with n states and m transitions. The running time of Algorithm 4.3 on input G is $O(m \cdot \log n)$.*

Exercise 4.14 {P2} [Mutual exclusion] Recall Peterson's mutual-exclusion protocol from Chapter 1. In the initial partition $\cong^I$, two states are equivalent iff they agree on all the observation predicates: $s \cong^I t$ iff $pc_1[s] = pc_1[t]$ and $pc_2[s] = pc_2[t]$. Construct the $\cong^I$-minimal quotient of G_{Pete} using first Algorithm 4.2 and then Algorithm 4.3. In both cases, show the intermediate results after each iteration of the while-loop. ■

4.3 Reachable Partition Refinement*

Consider a transition graph $G = (\Sigma, \sigma^I, \rightarrow)$ and an initial partition $\cong^I$. The $\cong^I$-minimal quotient is the graph $G/_{\min(\cong^I)}$ with state space $\Sigma/_{\min(\cong^I)}$ and initial states $\sigma^I/_{\min(\cong^I)}$. For verification, we need to compute only the reachable states of the $\cong^I$-minimal quotient. This suggests reformulating the partition refinement problem to account for reachability.

> MINIMAL REACHABLE QUOTIENT
>
> Let G be a transition graph and let $\cong$ be a G-partition. The *reachable stable partition* of $\cong$, denoted $\min^R(\cong)$, is the reachable region of the $\cong$-minimal quotient $G/_{\min(\cong)}$. The reachable subgraph of $G/_{\min(\cong)}$ is called the *$\cong$-minimal-reachable quotient*.

Remark 4.3 [Minimal reachable quotient] Let G be a transition graph with states Σ and reachable region σ^R. Let $\cong$ be a G-partition. The region σ^R need not be a block of $\min(\cong)$. The reachable stable partition $\min^R(\cong)$ is a partitioning of some region σ of G such that $\sigma^R \subseteq \sigma \subseteq \Sigma$. Thus, $\min^R(\cong)$ is not necessarily a G-partition, nor a refinement of $\cong$. A region τ in $\min^R(\cong)$ is contained in some $\cong$-equivalence class, and is stable with respect to every region in $\min^R(\cong)$. ■

To solve a reachability problem for the transition graphs, it suffices to construct the minimal-reachable quotient with respect to a suitably chosen initial partition.

Proposition 4.2 [Reachability] *Let G be a transition graph, $\cong$ be a G-partition, and σ be a block of $\cong$. Then, the answer to the reachability problem (G, σ) is* YES *iff $\sigma \cap \tau$ is nonempty for some $\tau \in \min^R(\cong)$.*

Algorithm 4.4 [Minimization with reachability]

Input: a transition graph $G = (\Sigma, \sigma^I, \to)$ and a G-partition $\cong^I$.
Output: the answer to the reachable-partition-refinement problem $(G, \cong^I)$.

$\cong := \cong^I$; $\sigma^R := \emptyset$
repeat
  {**assert** $\min(\cong^I)$ is a refinement of $\cong$ }
  $\cong_R := \{\sigma \in \cong \mid \sigma \cap \sigma^R \neq \emptyset\}$
  {**assert** $\sigma^R \subseteq post^*(\sigma^I)$, and for $\sigma \in \cong_R$, $|\sigma \cap \sigma^R| = 1$ }
  $U := \{\sigma \in \cong \setminus \cong_R \mid \sigma \cap (\sigma^I \cup post(\sigma^R)) \neq \emptyset\}$
  $V := \{(\tau, \upsilon) \in \cong_R \times \cong \mid \tau$ is unstable with respect to $\upsilon\}$

Search:	**or**	*Split:*
Choose $\sigma \in U$		Choose $(\tau, \upsilon) \in V$
Choose $s \in \sigma \cap (\sigma^I \cup post(\sigma^R))$		$\cong := Delete(\tau, \cong)$
$\sigma^R := Insert(s, \sigma^R)$		$\cong := \cong \cup Split(\tau, pre(\upsilon))$

  until $U = \emptyset$ and $V = \emptyset$
return $\cong_R$.

Figure 4.6: Simultaneous minimization and reachability

Reachable partition refinement

An instance $(G, \cong^I)$ of the *reachable-partition-refinement problem* consists of (1) a transition graph G and (2) [the *initial partition*] a G-partition $\cong^I$. The answer to the reachable-partition-refinement problem $(G, \cong^I)$ is the reachable stable partition $\min^R(\cong)$.

One possible solution to the reachable-partition-refinement problem is to first compute the $\cong^I$-minimal quotient and then analyze reachability. However, there are instances of the problem for which $\min(\cong^I)$ contains large, or even infinite, number of regions, but only a small number of them are reachable. Thus, the problem demands a solution that performs both the stabilization and reachability analysis simultaneously. An alternative strategy is shown in Figure 4.6.

As in the previous partition refinement algorithms, Algorithm 4.4 maintains a current partition $\cong$. The set σ^R contains states reachable from σ^I (at most one state per region of $\cong$). The set $\cong_R$ contains those regions of $\cong$ that are already known to be reachable. The algorithm computes the set U of regions that can be added to $\cong_R$ and the set V of unstable pairs of regions. A region σ belongs to U if it contains an initial state or a successor of a state already known to be reachable. A pair (τ, υ) belongs to the set V if τ is known to be

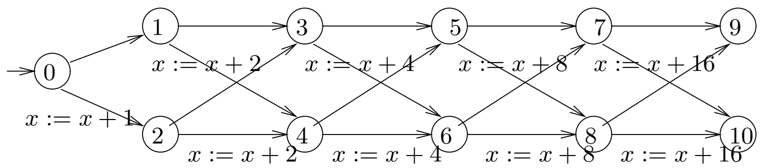

Figure 4.7: Example for computing minimal-reachable quotient

reachable, and is unstable with respect to υ. The algorithm either updates the reachability information for some region in U, or stabilizes some pair (τ, υ) in V by splitting τ. Thus searching is interleaved with stabilization in an arbitrary fashion. Stabilization involves splitting a reachable region τ with respect to $pre(\upsilon)$ for some $\cong$-equivalence class υ. Observe that a region is split only if it is known to be reachable. The algorithm terminates when neither search nor split is enabled. As in partition refinement, the coarsest stable partition $\min(\cong^I)$ is a refinement of the current partition $\cong$. Upon termination, $\cong_R$ is a subset of the coarsest stable partition $\min(\cong^I)$, and contains its reachable states. However, $\cong$ may contain unstable unreachable regions, and thus, need not equal $\min(\cong^I)$.

Theorem 4.6 [Minimization with reachability] *On an instance* $(G, \cong^I)$ *of reachable-partition refinement problem, if Algorithm 4.4 terminates, it outputs the reachable stable partition* $\min^R(\cong^I)$.

The size of σ^R, and hence, the number of regions in $\cong_R$, is nondecreasing, and is bounded by the number of regions in the output $\min^R(\cong^I)$. Every iteration either adds one more state to σ^R, or one more region to the partition $\cong$. It follows that if the coarsest stable refinement $\min(\cong^I)$ has finitely many regions, then Algorithm 4.4 is guaranteed to terminate. The algorithm may terminate even if $\min(\cong^I)$ has infinitely many regions. However, there are cases when $\min^R(\cong^I)$ has finitely many regions, and yet, the algorithm may execute forever. While the output does not depend upon the strategy used to choose between searching and splitting, the final partition $\cong$, and the number of iterations before termination, depend on the strategy.

Exercise 4.15 {P3} [Computing minimal-reachable quotient] Consider a symbolic transition graph with four boolean variables x, y, z, and w. The initial predicate is $x = \mathit{true} \wedge y = \mathit{false}$. The transition predicate is $(w' = x) \wedge (x' = \neg y) \wedge (y' = w \vee z)$. The initial partition contains two regions $[\![x \vee y]\!]$ and $[\![\neg x \wedge \neg y]\!]$. Compute the minimal-reachable quotient by executing Algorithm 4.4. How many regions does the output have? ■

Exercise 4.16 {T2} [Worst-case scenario for computing minimal-reachable quotient] Consider the symbolic transition graph shown in Figure 4.7. The graph has two variables, the location variable pc that ranges over the set $\{0 \ldots 10\}$, and a variable x that ranges over $\{0 \ldots 31\}$. The transitions are as shown. The assignments require that the updated value lies in the range $\{0 \ldots 31\}$ (e.g., the assignment $x := x+1$ stands for the guarded assignment $x < 31 \rightarrow x := x+1$). The initial predicate is $pc = 0 \wedge x = 0$. The initial partition $\cong^I$ contains one region $[\![pc = i]\!]$ per location $0 \leq i \leq 10$. How many regions does a $\cong^I$-minimal-reachable quotient have? Consider an execution of Algorithm 4.4, where splitting is preferred over searching. Show that, upon termination, for every value $0 \leq i \leq 31$ of x, the partition $\cong$ contains the singleton region $pc = 0 \wedge x = i$. ■

Lee-Yannakakis algorithm

The Lee-Yannakakis algorithm for constructing the minimal-reachable quotient modifies Algorithm 4.4 by imposing a deterministic strategy for searching and splitting. The algorithm is shown in Figure 4.8. The type of a graph partition is **partition**, and it supports insertion (*Insert*), deletion (*Delete*), enumeration (**foreach**), and the mapping *Find*.

Each iteration of the outer repeat-loop in Algorithm 4.5 consists of a searching phase, followed by the splitting phase. Search is performed until no more regions can be found reachable, thus, search has a priority over splitting.

As in Algorithm 4.4 the set σ^R contains reachable states, at most one per region of $\cong$. In the searching phase, the algorithm constructs the reachable regions $\cong_R$ by exploring the successors of states in σ^R. The set E contains the edges between the reachable regions. The search is performed in a depth first manner using the stack U.

The computation of the algorithm can be understood from the illustration of Figure 4.9. The partition contains 7 regions $\sigma_0, \ldots \sigma_6$. The regions σ_0, σ_1, σ_4 and σ_5 are found to be reachable in the searching phase. Each reachable region has a unique representative state in σ^R, for example, state s_0 for region σ_0. The reachability information is computed by considering the initial regions and by exploring successors of the representatives. Thus, at the end of the searching phase, we are guaranteed that the regions σ_2, σ_3 and σ_6 contain neither initial states nor successors of the representative states of the reachable regions.

In the splitting phase, the algorithm computes, for each reachable region σ, the subregion σ' that is stable with respect to the partition $\cong^{prev}$, and contains the reachable state $\sigma^R \cap \sigma$. Instead of splitting σ with respect to each region of $\cong^{prev}$, σ is split in at most two regions to avoid proliferation of regions.

To understand the splitting, reconsider the illustration of Figure 4.9. For the region σ_0, the algorithm computes the subregion σ'_0 (shown by the dotted lines) that contains states that agree with the representative s_0: for every state t in σ'_0, $post(t)$ intersects with σ_1 and σ_4, and does not intersect with σ_0, σ_2, σ_3, σ_5

Algorithm 4.5 [Lee-Yannakakis Algorithm]

Input: a transition graph $G = (\Sigma, \sigma^I, \rightarrow)$ and a G-partition $\cong^I$.
Output: the answer to the reachable-partition-refinement problem $(G, \cong^I)$.

local $\cong, \cong^{prev}$: **partition**; $\sigma^R, \sigma, \tau, \upsilon, \sigma'$: **region**; E: **set of region**$\times$**region**; s, t: **state**, U: **stack of state**
$\cong := \cong^I$; $\sigma^R := \mathit{EmptySet}$
foreach σ **in** $\cong^I$ **do**
  if $\sigma \cap \mathit{InitQueue}(G) \neq \emptyset$ **then** $\mathit{Insert}(\mathit{Element}(\sigma \cap \mathit{InitQueue}(G)), \sigma^R)$ **fi od**
repeat
Search:
  $U := \mathit{EmptyStack}$; $E := \mathit{EmptySet}$; $\cong_R := \mathit{EmptySet}$
  foreach s **in** σ^R **do**
    $U := \mathit{Push}(s, U)$; $\cong_R := \mathit{Insert}(\mathit{Find}(s, \cong), \cong_R)$ **od**
  while not $\mathit{EmptySet}(U)$ **do**
    $s := \mathit{Top}(U)$; $U := \mathit{Pop}(U)$; $\sigma := \mathit{Find}(s, \cong)$
    foreach t **in** $\mathit{PostQueue}(s, G)$ **do**
      $\tau := \mathit{Find}(t, \cong)$; $E := \mathit{Insert}((\sigma, \tau), E)$
      if not $\mathit{IsMember}(\tau, \cong_R)$ **then**
        $\sigma^R := \mathit{Insert}(t, \sigma^R)$; $U := \mathit{Push}(t, U)$;
        $\cong_R := \mathit{Insert}(\tau, \cong_R)$ **fi**
      od
    od
Split:
  $\cong^{prev} := \cong$
  foreach σ **in** $\cong_R$ **do**
    $\sigma' := \sigma$; $\tau := \mathit{PostQueue}(\sigma, G)$
    foreach (σ, υ) **in** E **do** $\sigma' := \sigma' \cap \mathit{PreQueue}(\upsilon, G)$; $\tau := \tau \backslash \upsilon$ **od**
    $\sigma' := \sigma' \backslash \mathit{pre}(\tau)$
    if $\sigma \neq \sigma'$ **then**
      $\cong := \mathit{Insert}(\sigma', \mathit{Insert}(\sigma \backslash \sigma', \mathit{Delete}(\sigma, \cong)))$
      if $\sigma \backslash \sigma' \cap \mathit{InitQueue}(G) \neq \emptyset$ **then**
        $\sigma^R := \mathit{Insert}(\mathit{Element}(\sigma \backslash \sigma'), \sigma^R)$ **fi fi**
    od
  od
until $\cong^{prev} = \cong$
return $\cong_R$.

Figure 4.8: Lee-Yannakakis algorithm for partition refinement

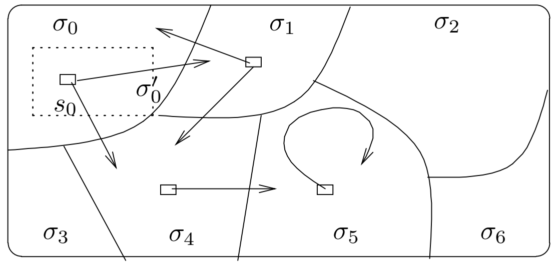

Figure 4.9: Computation of Lee-Yannakakis Algorithm

and σ_6. Clearly, σ_0' is nonempty as it contains s_0. Furthermore, σ_0' is known to be reachable with the representative state s_0. If it differs from σ_0, then σ_0 is split at the boundary of σ_0'. If the split part $\sigma_0 \backslash \sigma_0'$ contains an initial state, then it is declared reachable by choosing a representative state.

Lemma 4.5 [Stabilization in Lee-Yannakakis algorithm] *Let $\cong^{prev}$ be the value of the partition at the beginning of the splitting phase during some iteration of Algorithm 4.5. Let σ be a region in $\cong_R$, and let s be the unique state in $\sigma^R \cap \sigma$. Then, the subregion σ' computed at the end of the for-loop contains precisely those states t such that $t \rightarrow_G \tau$ iff $s \rightarrow_G \tau$ for all τ in $\cong^{prev}$.*

Exercise 4.17 {T3} [Stabilization in Lee-Yannakakis] Prove Lemma 4.5. ■

Once the subregion σ' is computed, the region σ is split at the boundary of σ'. The crucial aspect of the splitting strategy is that all regions are given a fair chance, in a round-robin order, to split.

Exercise 4.18 {P2} [Computing minimal-reachable quotient by Lee-Yannakakis] Execute Algorithm 4.5 on the input of Exercise 4.16. How many iterations are required before termination? ■

Suppose the reachable stable partition $\min^R(\cong^I)$ has n regions. During the execution of Algorithm 4.5, the number of regions that contain some reachable state of G is bounded by n. The convergence is established by the following lemma.

Lemma 4.6 [Convergence] *At the end of an iteration of the repeat-loop of Algorithm 4.5, the number of regions σ in $\cong$ with $\sigma \cap post_G^*(\sigma^I) \neq \emptyset$ either equals the number of regions in $\min^R(\cong^I)$, or exceeds the number of regions τ in $\cong^{prev}$ with $\tau \cap post_G^*(\sigma^I) \neq \emptyset$.*

Proof. Let $\upsilon = post^*_G(\sigma^I)$ be the reachable region of G. During the splitting phase each region σ in $\cong_R$ is split into two regions σ' and $\sigma\backslash\sigma'$ such that σ contains a reachable state and is stable with respect to each τ in $\cong^{prev}$ (Lemma 4.5). There are two cases to consider.

Case 1: for some σ in $\cong_R$, $\sigma\backslash\sigma'$ contains a state in υ. Then, the number of regions in the new partition containing reachable states exceeds the number of regions in the old partition containing reachable states.

Case 2: for all regions $\sigma \in\cong_R$, $\sigma\backslash\sigma'$ does not contain a state in υ. Let $\cong'_R$ be the set of regions σ'. We show that every region of the reachable stable partition $\min^R(\cong^I)$ is contained in a region of $\cong'_R$, and thus, the sets $\cong_R$, $\cong'_R$, and $\min^R(\cong^I)$ have the same cardinality.

First, we prove that every state in υ belongs to some σ'. We already know that, for all σ in $\cong_R$, $(\sigma\backslash\sigma') \cap \upsilon$ is empty. It suffices to show that for every region τ in $\cong^{prev} \setminus \cong_R$, $\tau \cap \upsilon$ is empty. Consider a region τ in $\cong^{prev} \setminus \cong_R$. Whenever a newly created region contains an initial state, one of its state is added to σ^R, and hence, the region gets added to $\cong_R$. Hence, $\tau \cap \sigma^I$ is empty. During the searching phase, all successors of all the states in σ^R are explored, and hence, $\tau \cap post(\sigma^R)$ is empty. Since every σ' in $\cong'_R$ is stable with respect to τ, and contains a state in σ^R, it follows that $\tau \cap post(\sigma')$ is empty for all σ' in $\cong'_R$. It follows that $\tau \cap \upsilon$ is empty.

For every σ' in $\cong'_R$, let $\sigma'' = \sigma' \cap \upsilon$. Consider two regions σ and τ in $\cong_R$. We know that σ' is stable with respect to τ. It follows that σ'' is also stable with respect to τ. Since $\sigma'' \subseteq \upsilon$, $post(\sigma'') \cap \tau = post(\sigma'') \cap \tau''$. It follows that σ'' is stable with respect to τ''. We conclude that the final output $\min^R(\cong^I)$ contains as many regions as $\cong_R$. ■

The running time of the algorithm depends upon the time complexity of the primitive operations on regions and partitions. If the number of successors of a state in G is bounded by k, then each searching phase requires at most kn operations. The number of operations during a splitting phase is bounded by the number of regions in $\cong_R$ and the number of edges in E. If the number of edges in the minimal-reachable quotient is m, then the size of E is bounded by m, and the splitting phase requires at most $m + n$ primitive operations. If $\min(\cong^I)$ has ℓ equivalence classes then Algorithm 4.5 is guaranteed to terminate after ℓ iterations.

Exercise 4.19 {T3} [Optimization of Lee-Yannakakis algorithm] The searching phase of Algorithm 4.5 builds the graph $(\cong_R, E)$ from scratch in each iteration. Suggest a modification so that the computation of one iteration is reused in the next. ■

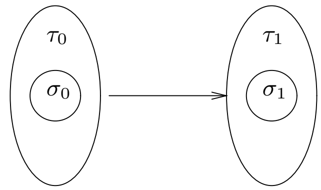

Figure 4.10: Reachable semistable quotient

Early termination

Lemma 4.6 ensures that the size of $\cong_R$ does not change after n iterations. Indeed, the graphs $(\cong_R, E)$ computed in searching phases are isomorphic after the n-th iteration. The splitting phase only removes subregions from each of the regions in $\cong_R$ without influencing the structure of the graph.

> REACHABLE SEMISTABLE QUOTIENT
> Let G be a transition graph and let $\cong^I$ be a G-partition. The pair $(\cong, E)$, for a set $\cong$ of regions of G and a set $E \subseteq \cong \times \cong$ of edges between the regions of $\cong$, is a *reachable* $\cong^I$*-semistable quotient* of G if (1) every region σ of the reachable stable partition $\min^R(\cong^I)$ is contained in a region $f(\sigma)$ of $\cong$, and (2) for two regions σ and τ of the reachable stable partition $\min^R(\cong^I)$, $\sigma \rightarrow_{\min(\cong^I)} \tau$ iff $(f(\sigma), f(\tau)) \in E$.

To understand the definition consider Figure 4.10 which shows a transition of a reachable semistable quotient of a transition graph G. Each region τ_i of a reachable semistable quotient contains a nonempty region σ_i that contains reachable states of G (the union of all σ_i equals the reachable region of G). The definition requires the transition from τ_0 to τ_1 to be stable with respect to the reachable subregions σ_i: from every state s in σ_0, $post_G(s) \cap \sigma_1$ is nonempty. However, there may be a state $s \in \tau_0 \backslash \sigma_0$ such that $post_G(s) \cap \tau_1$ is empty.

Example 4.6 [Semistable quotient] Consider the symbolic graph G with an integer variable x and a variable y that ranges over the interval $[0, 1]$ of real numbers. The initial predicate is $x = 0 \wedge y = 0$, and the transition predicate is

$$(x' = x + 1) \;\wedge\; (2y \leq 1 \rightarrow y' = 2y).$$

The initial partition $\cong^I$ contains the single region with all the states. In this case, $\min(\cong^I)$ has infinitely many regions. The reachable region is $[\![y = 0]\!]$, and contains infinitely many states. However, the minimal-reachable quotient is finite: for $\sigma = [\![y = 0]\!]$, the graph with the single state σ, and the single transition from σ to σ, is the minimal-reachable-quotient. There are infinitely many reachable semistable quotients. Executing Algorithm 4.5 for i iterations yields the semistable quotient that contains a single region $[\![y = 1/2^i]\!]$ with a single self-loop. ■

It follows that for a given transition graph and an initial partition, the corresponding reachable semistable partition is not uniquely defined. Two such reachable semistable partitions are isomorphic graphs. To solve a reachability problem for the transition graphs, it suffices to construct any reachable semistable quotient with respect to a suitably chosen initial partition.

Proposition 4.3 [Reachability and semistable quotients] *Let G be a transition graph, $\cong^I$ be a G-partition, and σ be a block of $\cong^I$. If $(\cong, E)$ is a reachable $\cong^I$-semistable quotient of G, then the answer to the reachability problem (G, σ) is* YES *iff $\sigma \cap \tau$ is nonempty for some τ in $\cong$.*

The Lee-Yannakakis algorithm is guaranteed to compute a reachable semistable quotient after linearly many iterations.

Theorem 4.7 [Computation of semistable partition] *Given an instance $(G, \cong^I)$ of reachable-partition-refinement, if the reachable stable partition $\min^R(\cong^I)$ has n regions, the pair $(\cong_R, E)$ at the end of the n-th iteration of the repeat-loop of Algorithm 4.5 is a reachable $\cong^I$-semistable quotient of G.*

Exercise 4.20 {T3} [Convergence to semistable quotient] Modify the proof of Lemma 4.6 to prove Theorem 4.7. ■

Since it suffices to compute a reachable semistable quotient to solve reachability problems, the execution of Algorithm 4.5 can be aborted, if there is a procedure that determines whether $(\cong_R, E)$ is a reachable semistable quotient. Observe that deciding whether $(\cong_R, E)$ is a reachable semistable quotient is an easier (static) problem compared to the dynamic problem of constructing one. While there are no general algorithms for this purpose, specialized solutions can be employed to exploit the structure of the update commands.

Exercise 4.21 {T4} [Cylinder-based refinement computation] Consider an instance $(P, \cong^I)$ of the reachable-partition-refinement problem with the following characteristics. The module P is a ruleset with a single enumerated variable x and k real-valued variables Y. Thus, the state-space Σ_P is the produce $\mathbb{T}x \times \mathbb{R}^k$. A *rational interval* I is a convex subset of $\mathbb{R}$ with rational endpoints. A region σ of P is *convex* if for all s and t in σ with $s(x) = t(x)$, for all $0 \leq \delta \leq 1$, the state u is also in σ, where $u(pc) = s(pc)$ and $u(y) = \delta \cdot s(y) + (1 - \delta) \cdot t(y)$ for all $y \in Y$. A region σ of P is a *cylinder* if there exist a value $m \in \mathbb{T}x$ and intervals I_y for variables $y \in Y$ such that a state s of P belongs to σ iff $s(x) = m$ and $s(y) \in I_y$ for $y \in Y$. Assume that the initial region σ_P^I is a cylinder, and every region in the initial partition $\cong^I$ is a cylinder. Furthermore, for every guarded assignment γ in the update command of P, the guard p_γ is a cylinder, and for all $y \in Y$, $e_\gamma^{y'}$ is of the form $az + b$ for some rational numbers a, b, and a variable $z \in Y$.

(1) Show that if a region σ of P is a cylinder, then $post_P(\sigma)$ is a finite union of cylinders, and $pre_P(\sigma)$ is a finite union of cylinders. (2) Show that every region in $\min(\cong^I)$ is convex. (3) Show that every region in $\min(\cong^I)$ is a cylinder. (4) Show that the problem of checking whether $(\cong, E)$ is a reachable $\cong^I$-semistable quotient can be formulated as a linear programming problem. What is the time-complexity of your test for semistability? ■

Appendix: Notation

Equivalences and partitions

A *partition* of a set A is a set of nonempty, pairwise disjoint subsets of A whose union is A. There is a one-to-one correspondence between the equivalences on A and the partitions of A. Given an equivalence $\cong$ on A and an element $a \in A$, we write $a/_{\cong}$ for the $\cong$-*equivalence class* $\{b \in A \mid b \cong a\}$ of a. The set $A/_{\cong}$ of $\cong$-equivalence classes is a partition of A. In this way, each partition of A is induced by a unique equivalence on A. Therefore, whenever we refer to a partition of A, we use a notation like $A/_{\cong}$, which indicates the corresponding equivalence. We also freely attribute properties and derivatives of equivalences to partitions, and vice versa.

Let $\cong$ be an equivalence on A. The equivalence $\cong$ is *finite* if $\cong$ has finitely many equivalence classes. A union of $\cong$-equivalence classes is called a *block* of $\cong$. If $\cong$ is finite with n equivalence classes, then $\cong$ has 2^n blocks. Given two equivalences $\cong_1$ and $\cong_2$ on A, the equivalence $\cong_1$ *refines* the equivalence $\cong_2$, written $\cong_1 \preceq \cong_2$, if $a \cong_1 b$ implies $a \cong_2 b$. If $\cong_1$ refines $\cong_2$, then every block of $\cong_2$ is a block of $\cong_1$. For a set E of equivalence relations on A, the *join* $\bigcup^* E$ is the transitive closure of the union $\bigcup E$ of the relations in E; the join $\bigcup^* E$ is an equivalence on A. The refinement relation $\preceq$ is a complete lattice on the set of equivalences on A. The least upper $\preceq$-bound for a set E of equivalences on A is the join $\bigcup^* E$; the greatest lower $\preceq$-bound for E is the intersection $\bigcap E$.

Exercise 4.22 {} [Partition theorems] Prove all claims made in the previous paragraph. ■

Groups

A *goup* is a set A with a binary function $\circ \colon A^2 \mapsto A$, called the multiplication operation, such that (1) $\circ$ is associative, (2) there exists an element that is identity for $\circ$, and (3) every element of A has an inverse with respect to $\circ$. Consider a group $(A, \circ)$. A *subgroup* of A is a subset $B \subseteq A$ such that $(B, \circ)$ is a group. For a subset $B \subseteq A$, the subgroup generated by B, denoted $closure(B)$, is the smallest subgroup of $(A, \circ)$ that contains B. The elements in B are called *generators* for the group $closure(B)$.