Automatic Generation of Invariants and Intermediate Assertions ⋆

Nikolaj Bjørner, Anca Browne and Zohar Manna [a]

[a] *Computer Science Department*
Stanford University
Stanford, CA 94305
{nikolaj,anca,zm}@cs.stanford.edu

Abstract

Verifying temporal specifications of reactive and concurrent systems commonly relies on generating auxiliary assertions and on strengthening given properties of the system. This can be achieved by two dual approaches: The *bottom-up* method performs an abstract forward propagation (computation) of the system, generating auxiliary assertions; the *top-down* method performs an abstract backward propagation to strengthen given properties. *Exact* application of these methods is complete but is usually infeasible for large-scale verification. *Approximation* techniques are often needed to complete the verification.

We give an overview of known methods for generation of auxiliary invariants in the verification of invariance properties. We extend these methods, by formalizing and analyzing a general verification rule that uses *assertion graphs* to generate auxiliary assertions for the verification of general safety properties.

1 Introduction

The deductive verification of temporal specifications of reactive and concurrent systems commonly relies upon finding suitable auxiliary assertions and strengthened properties [MP95]. This paper describes a general framework for

⋆ This research was supported in part by the National Science Foundation under grant CCR-92-23226, the Advanced Research Projects Agency under NASA grant NAG2-892, the United States Air Force Office of Scientific Research under grant F49620-93-1-0139, and the Department of the Army under grant DAAH04-95-1-0317.

generating invariants and intermediate assertions that can help in the verification of general temporal safety properties. General temporal safety formulas can express properties such as invariance, first-in-first-out ordering, causality, and bounded overtaking.

Deductive methods provide a verification rule that proves invariance properties by establishing first-order premises. These premises often require an auxiliary invariant which strengthens the invariance property being proved. There are two ways to find the strengthed invariant: forward propagation (bottom-up method) is a symbolic forward execution of the system $\mathcal{S}$ yielding an invariant that characterizes the set of reachable states; backward propagation (top-down method) is a symbolic backward execution of $\mathcal{S}$ from the states satisfying the invariance property being proved, yielding an invariant that characterizes the states that maintain the invariance property. We will see that both methods, in principle, generate adequate auxiliary invariants. Early work on forward and backward propagation in sequential program verification includes [GW75,KM76]. More recently [MP95,BLS96] have extended the methodologies to concurrent systems.

This paper extends traditional methodologies for invariance properties by formulating a new inference rule and analyzing it to obtain new methods for general safety formulas. Our main tool is the use of *assertion graphs* that summarize the models of safety formulas. The exposition demonstrates a tight connection between the verification rule and the duality of backward and forward propagation.

Exact forward or backward propagation of a system $\mathcal{S}$ may not terminate when the state space of $\mathcal{S}$ is infinite or unmanageably large. This motivates the study of approximate, but terminating, propagation methods over different domains. General safety formulas were first treated in [BBM95] where techniques from linear algebra, linear programming and monadic second-order logic were applied in several case studies of approximate propagation. Linear algebra has long been applied to the automatic discovery of linear equalities between system variables [Lau73,Kar76]. Linear programming was proposed in [CH78] to express linear constraints among system variables in the form of convex polyhedra. It is also one of the most prominent examples of the abstract interpretation theory introduced in [CC77]. Real-time systems can be analyzed using successive over- and under-approximations of propagation with convex polyhedra [DW95]. Monadic second-order logic applied in set-based program analysis [Hei92] provides unary constraints on the values of program variables. The case studies used in this paper overview some of these techniques while extending them to handle parameterized systems. In particular, we show how polyhedral analysis can be extended both with respect to parameterization and general safety properties. The use of approximation domains appeals in general naturally to techniques developed in constraint programming.

The paper is structured as follows. Section 2 summarizes preliminary notions such as the computational model of reactive systems, linear-time temporal logic and fixedpoints. In Section 3 we present the verification rule and introduce bottom-up and top-down methods for invariance properties. The methods are extended to handle general safety properties in Section 4. Section 5 introduces approximation techniques to support the exact methods presented in Sections 3 and 4. In the same section, approximation is used to establish invariance and general safety properties for examples where exact propagation would fail.

Some of the methods described in this paper are included in a released version of the verification system STeP, the *Stanford Temporal Prover* [BBC+95]. STeP verifies linear-time temporal logic specifications of reactive and concurrent systems, combining algorithmic and deductive techniques.

2 Preliminaries

2.1 Transition Systems

Following [MP95], our computational model for reactive systems is that of a *transition system*, $\mathcal{S} = \langle \mathcal{V}, \Theta, \mathcal{T} \rangle$, where $\mathcal{V}$ is a finite set of system variables, Θ is the initial condition, and $\mathcal{T}$ is a finite set of transitions. Fairness assumptions are ignored when establishing safety properties, so the usual sets of just and compassionate transitions are omitted. The vocabulary $\mathcal{V}$ contains data variables, control variables and auxiliary variables. The set of *states* (interpretations) over $\mathcal{V}$ is denoted by Σ. We assume a first-order assertion language $\mathcal{A}$ over $\mathcal{V}$. The initial condition Θ is an assertion in this language. A transition τ maps each state $s \in \Sigma$ into a (possibly empty) set of τ-successor states, $\tau(s) \subseteq \Sigma$. The mapping associated with τ is defined by an assertion $\rho_\tau(\overline{x}, \overline{x}')$, called the *transition relation*, which relates the values $\overline{x}$ of the variables in state s and the values $\overline{x}'$ in a successor state $s' \in \tau(s)$. We require that $\mathcal{T}$ contains a transition τ_I, called the *idling transition*, such that $\tau(s) = \{s\}$ for every state s.

A *computation* of a system $\mathcal{S}$ is an infinite sequence of states $s_0, s_1, s_2, \ldots$, such that s_0 is an initial state satisfying Θ and for every $i \geq 0$ there is a transition $\tau \in \mathcal{T}$ satisfying $s_{i+1} \in \tau(s_i)$.

To facilitate the representation of systems, some of our examples are given in SPL (Simple Programming Language), which is used to represent concurrent programs in [MP95,BBC+95]. SPL statements are translated into transitions in a straightforward manner. For example, the assignment statement

$$\ell_0 : x := y + 1;\ \ell_1 :$$

assigns $y + 1$ to x when control resides at location ℓ_0, and simultaneously moves control to ℓ_1. The transition τ corresponding to this statement has the transition relation

$$\rho_\tau(\ell_0, \ell_1, x, y, \ell_0', \ell_1', x', y') : \ \ at\_\ell_0 \ \wedge\ x' = y + 1 \ \wedge\ y' = y \ \wedge\ at\_\ell_1' \ \wedge\ \neg at\_\ell_0' \ .$$

2.2 Preconditions and Postconditions

The weakest precondition $\textsc{wp}(\tau, \varphi)(\overline{x})$ and strongest postcondition $\textsc{sp}(\tau, \varphi)(\overline{x})$ of an assertion $\varphi(\overline{x})$, relative to a transition τ, are defined by

$$\begin{aligned}
\textsc{wp}(\tau, \varphi)(\overline{x}) &\ \triangleq\ \forall \overline{x}' \cdot \rho_\tau(\overline{x}, \overline{x}') \rightarrow \varphi(\overline{x}') \\
\textsc{sp}(\tau, \varphi)(\overline{x}) &\ \triangleq\ \exists \overline{x}_0 \cdot \rho_\tau(\overline{x}_0, \overline{x}) \wedge \varphi(\overline{x}_0) \ .
\end{aligned}$$

$\textsc{wp}$ characterizes the states that must reach a φ-state (i.e., a state satisfying φ) by taking τ. $\textsc{sp}$ characterizes the states reachable from a φ-state by taking τ.

For example, transitions associated with guarded assignments of the form **if** $c(\overline{x})$ **then** $\overline{x} := e(\overline{x})$ have transition relations of the form $\rho_\tau(\overline{x}, \overline{x}') : c(\overline{x}) \ \wedge\ \overline{x}' = e(\overline{x})$. Their weakest precondition, $\textsc{wp}(\tau, \varphi)(\overline{x}) :\ \forall \overline{x}'.c(\overline{x}) \ \wedge\ \overline{x}' = e(\overline{x}) \rightarrow \varphi(\overline{x}')$, can be simplified to $\textsc{wp}(\tau, \varphi)(\overline{x}) :\ c(\overline{x}) \ \rightarrow\ \varphi(e(\overline{x}))$.

The two operators are monotone in φ since all occurrences of φ are positive, i.e., they are under the scope of an even number of negations. Thus,

$$(\varphi \rightarrow \psi) \ \rightarrow\ (\textsc{wp}(\tau, \varphi) \rightarrow \textsc{wp}(\tau, \psi)), \qquad (\varphi \rightarrow \psi) \ \rightarrow\ (\textsc{sp}(\tau, \varphi) \rightarrow \textsc{sp}(\tau, \psi)).$$

Also,

$$\begin{array}{ll}
\textsc{wp}(\tau, \varphi \wedge \psi) \leftrightarrow \textsc{wp}(\tau, \varphi) \wedge \textsc{wp}(\tau, \psi) & \quad \textsc{sp}(\tau, \varphi \vee \psi) \leftrightarrow \textsc{sp}(\tau, \varphi) \vee \textsc{sp}(\tau, \psi) \\
\textsc{wp}(\tau, \varphi \vee \psi) \leftarrow \textsc{wp}(\tau, \varphi) \vee \textsc{wp}(\tau, \psi) & \quad \textsc{sp}(\tau, \varphi \wedge \psi) \rightarrow \textsc{sp}(\tau, \varphi) \wedge \textsc{sp}(\tau, \psi).
\end{array}$$

We will also use the notation

$$\textsc{wp}(\mathcal{T}, \varphi) \ \triangleq\ \textstyle\bigwedge_{\tau \in \mathcal{T}} \textsc{wp}(\tau, \varphi) \qquad \textsc{sp}(\mathcal{T}, \varphi) \ \triangleq\ \textstyle\bigvee_{\tau \in \mathcal{T}} \textsc{sp}(\tau, \varphi) \ .$$

Notice that

$$\varphi \rightarrow \text{WP}(\mathcal{T}, \psi) \quad \text{iff} \quad \text{SP}(\mathcal{T}, \varphi) \rightarrow \psi \ .$$

2.3 Linear-Time Temporal Logic

We specify properties of reactive systems using linear-time temporal logic. Linear-time temporal logic allows us to specify behaviours over the set of all computations of a system. A *temporal formula* is constructed from state formulas (assertions), which are formulas from the first-order assertion language $\mathcal{A}$. To state formulas we apply boolean connectives (such as $\neg$, $\wedge$, $\vee$, $\rightarrow$ in order of falling precedence), quantifiers ($\forall$, $\exists$) and temporal operators.

The temporal operators used in this paper are future operators $\Box$ (*always in the future*), $\mathcal{W}$ (*waiting-for, unless*), $\bigcirc$ (*next*), and their past counterparts $\boxminus$ (*always in the past*), $\mathcal{B}$ (*back-to*) and $\ominus$ (*previously*).

A *model* for a temporal formula φ is an infinite sequence of states $\sigma : s_0, s_1, s_2, \ldots$, where each state s_j provides an interpretation for the variables occurring in φ. Given a model σ, we present an inductive definition of the notion of φ holding at position j, $j \geq 0$, in σ, denoted by $\langle \sigma, j \rangle \vDash \varphi$:

$$\begin{array}{lcll}
\langle \sigma, j \rangle \vDash \varphi & \Longleftrightarrow & s_j \vDash \varphi & \text{if } \varphi \in \mathcal{A} \\
 & & \multicolumn{2}{l}{\text{That is, } \varphi \text{ is evaluated locally, using the interpretation in } s_j} \\
\langle \sigma, j \rangle \vDash \bigcirc \varphi & \Longleftrightarrow & \langle \sigma, j+1 \rangle \vDash \varphi & \\
\langle \sigma, j \rangle \vDash \Box \varphi & \Longleftrightarrow & \forall j' : j' \geq j \cdot \langle \sigma, j' \rangle \vDash \varphi & \\
\langle \sigma, j \rangle \vDash \varphi \, \mathcal{W} \, \psi & \Longleftrightarrow & \forall j' : j' \geq j \cdot \langle \sigma, j' \rangle \vDash \varphi \ \text{ or} & \\
 & & \exists k : k \geq j \cdot \langle \sigma, k \rangle \vDash \psi \text{ and } \forall j' : j \leq j' < k \cdot \langle \sigma, j' \rangle \vDash \varphi & \\
\langle \sigma, j \rangle \vDash \ominus \varphi & \Longleftrightarrow & j > 0 \ \text{ and } \ \langle \sigma, j-1 \rangle \vDash \varphi & \\
\langle \sigma, j \rangle \vDash \boxminus \varphi & \Longleftrightarrow & \forall j' : 0 \leq j' \leq j \cdot \langle \sigma, j' \rangle \vDash \varphi & \\
\langle \sigma, j \rangle \vDash \varphi \, \mathcal{B} \, \psi & \Longleftrightarrow & \forall j' : 0 \leq j' \leq j \cdot \langle \sigma, j' \rangle \vDash \varphi \ \text{ or} & \\
 & & \exists k : 0 \leq k \leq j \cdot \langle \sigma, k \rangle \vDash \psi \text{ and } \forall j' : j \geq j' > k \cdot \langle \sigma, j' \rangle \vDash \varphi \ . &
\end{array}$$

Other temporal connectives are defined as abbreviations, e.g., $\Diamond\varphi = \neg \Box \neg\varphi$, $\diamondminus \varphi = \neg \boxminus \neg\varphi$.

A temporal formula φ is $\mathcal{S}$-*valid*, denoted $\mathcal{S} \vDash \varphi$, if for each computation σ of $\mathcal{S}$, $\langle \sigma, 0 \rangle \vDash \varphi$. A state s is said to be an $\mathcal{S}$-*reachable* state if it can be reached by some computation of $\mathcal{S}$. A state formula φ is $\mathcal{S}$-*state-valid* if $s \vDash \varphi$ for every $\mathcal{S}$-reachable state s of $\mathcal{S}$. See [MP95] for a more detailed discussion of linear-time temporal logic and reactive systems.

2.4 Fixedpoints

For a monotone operator $G : \mathcal{A} \mapsto \mathcal{A}$, such as WP and SP, we write the least fixedpoint as $\mu X.G(X)$ and the greatest fixedpoint of G as $\nu X.G(X)$. To give meaning to the fixedpoint operators μ and ν we introduce *denotations* $[\![\varphi]\!]_\rho$ of assertions φ in $\mathcal{A}$, where ρ maps free predicate symbols to subsets of Σ. By $[\![\varphi]\!]_\rho$ we will understand the set of states $s \in \Sigma$, such that φ holds in state s when free predicate symbols are interpreted by ρ; in short: $[\![\varphi]\!]_\rho = \{s \in \Sigma \mid \rho, s \vDash \varphi\}$, In particular $[\![G(X)]\!]_{\rho[S/X]}$ is the denotation of $G(X)$ where the modified substitution $\rho[S/X]$ maps the free predicate symbol X to S and behaves like ρ for every other predicate. The denotations of the fixedpoint operators are now defined as

$$\begin{aligned} [\![\mu X.G(X)]\!] &\triangleq \bigcap\left\{S \subseteq \Sigma \mid [\![G(X)]\!]_{[S/X]} \subseteq S\right\} \\ [\![\nu X.G(X)]\!] &\triangleq \bigcup\left\{S \subseteq \Sigma \mid S \subseteq [\![G(X)]\!]_{[S/X]}\right\} \ . \end{aligned}$$

The monotonicity of G ensures that $[\![\mu X.G(X)]\!]$ is the unique least fixedpoint, i.e., the least set $S \subseteq \Sigma$ such that $S = [\![G(X)]\!]_{[S/X]}$, and that $[\![\nu X.G(X)]\!]$ is the unique greatest fixedpoint.

Searching for fixedpoints in the assertion language $\mathcal{A}$ itself, we apply G repeatedly starting from F (false). If the ascending chain F, $G(\mathrm{F})$, $G^{(2)}(\mathrm{F})$, ... converges in a finite number of iterations, its limit is an assertion in $\mathcal{A}$ equivalent to $\mu X.G(X)$. Similarly, we can generate the descending chain $\mathrm{T}, G(\mathrm{T}), G^{(2)}(\mathrm{T}), \ldots$, starting from T (true); if it converges an assertion in $\mathcal{A}$ is produced which is equivalent to $\nu X.G(X)$. In general, the generated sequences are not guaranteed to converge to the fixedpoints in a finite number of steps.

3 Invariance

An assertion p is $\mathcal{S}$-*invariant* (*invariant* for short) if $\mathcal{S} \vDash \Box\, p$. To establish that a given assertion p is $\mathcal{S}$-invariant, we use the verification rule INV (Figure 1). This sound and relatively complete[1] proof rule reduces the verification of $\Box\, p$ to first-order premises. For a given transition system $\mathcal{S}$ and assertion p, to prove that $\Box\, p$ is $\mathcal{S}$-valid, we have to find a strengthened assertion φ such that the first-order premises $I1{-}I3$ are $\mathcal{S}$-state-valid. In premise $I3$ we

[1] Completeness is here understood relative to the expressiblity of the first-order language.

$$
\begin{array}{ll}
\text{For} & \text{transition system } \mathcal{S} : \langle \mathcal{V}, \Theta, \mathcal{T} \rangle, \text{ and} \\
& \text{assertions } p \text{ and } \varphi
\end{array}
$$

$$
\begin{array}{ll}
I1 & \Theta \rightarrow \varphi \\
I2 & \varphi \rightarrow p \\
I3 & \{\varphi\}\ \mathcal{T}\ \{\varphi\} \\
\hline
\multicolumn{2}{c}{\mathcal{S} \models \Box\, p}
\end{array}
$$

Fig. 1. Rule INV (invariance)

use the notation $\{\varphi\}\ \mathcal{T}\ \{\psi\} \triangleq \bigwedge_{\tau \in \mathcal{T}} \{\varphi\}\tau\{\psi\}$ where each *verification condition* $\{\varphi\}\tau\{\psi\}$ can be expressed equivalently in any of the following forms

$$
\begin{array}{ll}
\text{standard:} & \forall \overline{x} \forall \overline{x}' \cdot [\varphi(\overline{x}) \ \wedge\ \rho_\tau(\overline{x}, \overline{x}') \rightarrow \psi(\overline{x}')] \\
\text{weakest precondition:} & \forall \overline{x} \cdot [\varphi(\overline{x}) \ \rightarrow\ \mathrm{WP}(\tau, \psi)(\overline{x})] \\
\text{strongest postcondition:} & \forall \overline{x}' \cdot [\mathrm{SP}(\tau, \varphi)(\overline{x}') \ \rightarrow\ \psi(\overline{x}')] \ .
\end{array}
$$

When establishing premises $I1 - I3$ of rule INV it is sound to assert any previously established invariant ψ as an axiom. This will be used throughout the paper.

The main difficulty in using rule INV is finding the strengthened assertion φ. We now show that the strongest and weakest candidates for φ can be given fixedpoint characterizations.

3.1 Forward propagation

Define the operator $\mathcal{F}$ by

$$
\mathcal{F}(X) \quad \triangleq \quad \Theta \vee \mathrm{SP}(\mathcal{T}, X) \ .
$$

A formula φ satisfies $I1$ and $I3$ iff $\mathcal{F}(\varphi) \rightarrow \varphi$. Since $\mathcal{F}$ is monotone, i.e., if $X_1 \rightarrow X_2$ then $\mathcal{F}(X_1) \rightarrow \mathcal{F}(X_2)$, the fixedpoint formula

$$
\varphi_{\mathcal{F}} : \quad \mu X . \mathcal{F}(X)
$$

provides the strongest assertion φ satisfying $I1$ and $I3$. We therefore have

$$
\text{(F1)} \quad \Theta \rightarrow \varphi_{\mathcal{F}} \qquad\qquad \text{(F2)} \quad \mathrm{SP}(\mathcal{T}, \varphi_{\mathcal{F}}) \rightarrow \varphi_{\mathcal{F}} \ .
$$

Notice that $\varphi_{\mathcal{F}}$ precisely characterizes the set of reachable states:

$$\text{state } s \text{ is } \mathcal{S}\text{-accessible} \quad \text{iff} \quad s \vDash \varphi_{\mathcal{F}} \ .$$

The implication from left to right holds for any formula satisfying $I1$ and $I3$; the converse holds because $\varphi_{\mathcal{F}}$ is the strongest such formula.

As $\mathcal{F}$ is monotone, if the sequence starting from F (false)

$$\underbrace{\mathrm{F}}_{\varphi_0} \ \rightarrow \ \underbrace{\mathcal{F}(\varphi_0)}_{\varphi_1} \ \rightarrow \ \underbrace{\mathcal{F}(\varphi_1)}_{\varphi_2} \rightarrow \cdots$$

converges in finitely many steps, i.e., $\varphi_n \leftrightarrow \varphi_{n+1}$ is $\mathcal{S}$-state-valid for some n, then its limit (φ_n) is $\varphi_{\mathcal{F}}$.

3.2 Backward propagation

Given an assertion p, define the monotone operator $\mathcal{B}$ by

$$\mathcal{B}(Y) \quad \triangleq \quad p \wedge \mathrm{WP}(\mathcal{T}, Y) \ .$$

A formula satisfies satisfies $I2$ and $I3$ iff $\varphi \rightarrow \mathcal{B}(\varphi)$. The greatest fixedpoint

$$\varphi_{\mathcal{B}} : \ \nu Y.\mathcal{B}(Y)$$

provides the weakest φ satisfying $I2$ and $I3$. We therefore have

$$\text{(B1)} \quad \varphi_{\mathcal{B}} \rightarrow p \qquad\qquad \text{(B2)} \quad \varphi_{\mathcal{B}} \rightarrow \mathrm{WP}(\mathcal{T}, \varphi_{\mathcal{B}}) \ .$$

$\varphi_{\mathcal{B}}$ precisely characterizes the *p-invariant states* i.e., the states where every computation of $\mathcal{S}$ starting in s satisfies $\Box\, p$.

Since $\mathcal{B}$ is monotone, if the sequence starting from T (true)

$$\underbrace{\mathrm{T}}_{\varphi_0} \ \leftarrow \ \underbrace{\mathcal{B}(\varphi_0)}_{\varphi_1} \ \leftarrow \ \underbrace{\mathcal{B}(\varphi_1)}_{\varphi_2} \ \leftarrow \ \cdots$$

converges in finitely many steps, then its limit is $\varphi_{\mathcal{B}}$.

3.3 The forward/backward duality

Abstract state space exploration from the set of initial states, also called forward propagation (*bottom-up analysis*), does not depend on the system property p we want to establish. Backward propagation (*top-down analysis*), on the other hand, explores the state space starting with the p-states.

Since $\varphi_{\mathcal{F}}$ captures exactly the reachable state space, whereas $\varphi_{\mathcal{B}}$ only collects enough information to establish p, one can prove that the following are equivalent:

$$1\colon \Theta \to \varphi_{\mathcal{B}}, \quad 2\colon \varphi_{\mathcal{F}} \to p, \quad 3\colon \varphi_{\mathcal{F}} \to \varphi_{\mathcal{B}}, \quad 4\colon p \text{ is } \mathcal{S}\text{-invariant.}$$

The correspondence is best illustrated by the commuting diagram:

$$\begin{tikzcd} \Theta \arrow[r, "(\mathrm{F1})"] \arrow[d] & \varphi_{\mathcal{F}} \arrow[d] \arrow[dl] \\ \varphi_{\mathcal{B}} \arrow[r, "(\mathrm{B1})"'] & p \end{tikzcd}$$

where the horizontal implications are given by the fixedpoint equations. If one of the downwards directed implications is present, all the others must also be.

The diagram suggests that if p is an invariant, then the $\varphi_{\mathcal{F}}$ states are a subset of the $\varphi_{\mathcal{B}}$ states as reflected in Figure 2.

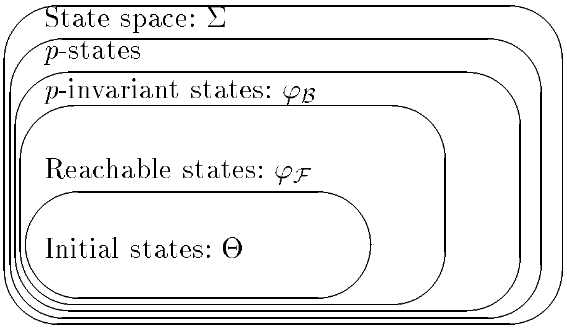


Fig. 2. State space.

As Figure 2 shows, the p-invariant states, given by $\varphi_{\mathcal{B}}$, are a subset of the p-states, which are those states where p holds but is not necessarily preserved by the transitions.

3.4 An example of forward and backward propagation

We analyze program BAKERY (Figure 3) using both forward and backward propagation. The program guarantees mutual exclusion, that is, ℓ_3 and m_3 are never reached at the same time. Synchronization is provided by the integer variables y_1 and y_2, which can be thought of as numbers used in waiting-lines at bakeries.

$$\textbf{local}\ \ y_1, y_2 : \textbf{integer where}\ y_1 = y_2 = 0$$

$$P_1 :: \left[\begin{array}{l}\textbf{loop forever do}\\ \quad\left[\begin{array}{ll}\ell_0: & \textbf{noncritical}\\ \ell_1: & y_1 := y_2 + 1\\ \ell_2: & \textbf{await}\ (y_2 = 0 \lor\\ & \qquad\qquad y_1 \leq y_2)\\ \ell_3: & \textbf{critical}\\ \ell_4: & y_1 := 0\end{array}\right]\end{array}\right] \;||\; P_2 :: \left[\begin{array}{l}\textbf{loop forever do}\\ \quad\left[\begin{array}{l}m_0 : \textbf{noncritical}\\ m_1 : y_2 := y_1 + 1\\ m_2 : \textbf{await}\ (y_1 = 0 \lor\\ \qquad\qquad y_2 < y_1)\\ m_3 : \textbf{critical}\\ m_4 : y_2 := 0\end{array}\right]\end{array}\right]$$

Fig. 3. Program BAKERY (Program BAKERY for mutual exclusion)

Forward propagation The method requires computing the sequence

$$\underbrace{\mathrm{F}}_{\varphi_0} \rightarrow \underbrace{\mathcal{F}(\varphi_0)}_{\varphi_1} \rightarrow \underbrace{\mathcal{F}(\varphi_1)}_{\varphi_2} \rightarrow \cdots$$

until a limit is found.

Figure 4 represents the iterations of $\mathcal{F}$ as layers in a directed graph, growing bottom-up from the initial condition $\varphi_1 :\ \Theta$. The i-th iteration of $\mathcal{F}$, represented by φ_i, is the disjunction of the nodes that are reachable from the source on a path of at most depth $i-1$.

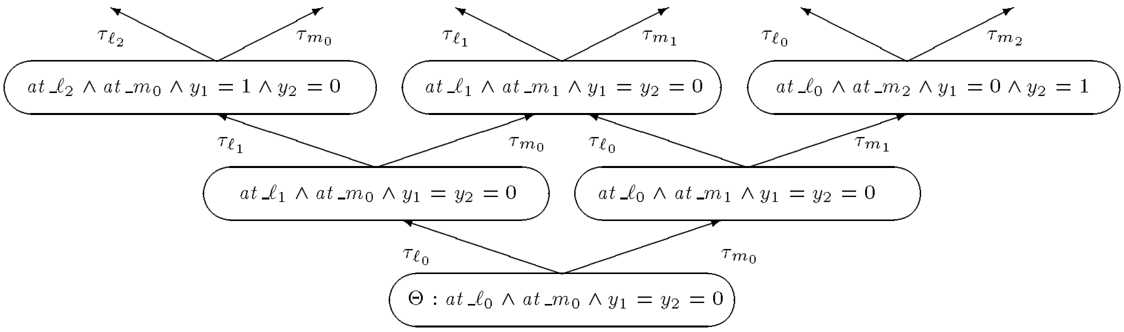


Fig. 4. Forward propagation

From the initial condition Θ we calculate

$$\begin{aligned}
\text{SP}(\mathcal{T},\Theta) &= \textstyle\bigvee_{\tau\in\mathcal{T}}\text{SP}(\tau,\Theta)\\
&= \text{SP}(\tau_{\ell_0},\Theta)\vee\text{SP}(\tau_{m_0},\Theta)\\
&= (at_\ell_1 \ \wedge\ at_m_0 \ \wedge\ y_1=y_2=0) \quad\vee\quad (at_\ell_0 \ \wedge\ at_m_1 \ \wedge\ y_1=y_2=0)
\end{aligned}$$

which generates the two disjuncts just above Θ. An additional iteration of $\mathcal{F}$ generates the three disjuncts at the third level. The sequence $\varphi_1,\varphi_2,\varphi_3,\ldots$ does not converge to a fixedpoint in a finite number of steps, since $\varphi_{5k}\wedge at\_\ell_2\wedge at\_m_2$ is equivalent to

$$at\_\ell_2 \ \wedge\ at\_m_2 \ \ \wedge\ |y_1-y_2|=1 \ \wedge\ 1\le y_1\le k+1 \ \wedge\ 1\le y_2\le k+1 \ .$$

However, it is possible to express the strongest invariant (representing the reachable state space) of the program by the assertion:

$$\begin{aligned}
\varphi_{\mathcal{F}}:\ & [at_\ell_{0,1}\wedge at_m_{0,1}\wedge y_1=0\wedge y_2=0]\\
\vee\ & [at_\ell_{0,1}\wedge at_m_{2,3,4}\wedge y_1=0\wedge y_2\ge 1]\\
\vee\ & [at_\ell_{2,3,4}\wedge at_m_{0,1}\wedge y_1\ge 1\wedge y_2=0]\\
\vee\ & [at_\ell_2\wedge at_m_2\wedge y_1\ge 1\wedge y_2\ge 1\wedge |y_1-y_2|=1]\\
\vee\ & [at_\ell_2\wedge at_m_{3,4}\wedge y_2\ge 1\wedge y_1-y_2=1]\\
\vee\ & [at_\ell_{3,4}\wedge at_m_2\wedge y_1\ge 1\wedge y_2-y_1=1]\ .
\end{aligned}$$

Backward propagation Backward propagation starts from an invariant candidate, in this case

$$\Box\neg(at\_\ell_3 \ \wedge\ at\_m_3),$$

which expresses mutual exclusion in the critical sections.

We compute the terms of the sequence

$$\underbrace{\mathrm{T}}_{\varphi_0} \ \leftarrow\ \underbrace{\mathcal{B}(\varphi_0)}_{\varphi_1} \ \leftarrow\ \underbrace{\mathcal{B}(\varphi_1)}_{\varphi_2} \ \leftarrow\ \cdots$$

until a limit is found. Applying $\mathcal{B}$ once generates $\varphi_1:\neg(at\_\ell_3 \ \wedge\ at\_m_3)$. In the second iteration of $\mathcal{B}$ we calculate:

$$\begin{aligned}
\text{WP}(\mathcal{T},\varphi_1) &= \textstyle\bigwedge_{\tau\in\mathcal{T}}\text{WP}(\tau,\varphi_1)\\
&= \text{WP}(\tau_{\ell_2},\varphi_1)\ \wedge\ \text{WP}(\tau_{m_2},\varphi_1)\\
&= (at_\ell_2\wedge at_m_3\rightarrow y_2\ne 0\wedge y_1>y_2)\\
&\quad\wedge\ (at_\ell_3\wedge at_m_2\rightarrow y_1\ne 0\wedge y_1\le y_2).
\end{aligned}$$

Continuing mechanically in this fashion we obtain the formulas shown in Figure 5. By calculating $\text{WP}(\tau,\varphi)$, where τ labels an edge pointing to a φ-node,

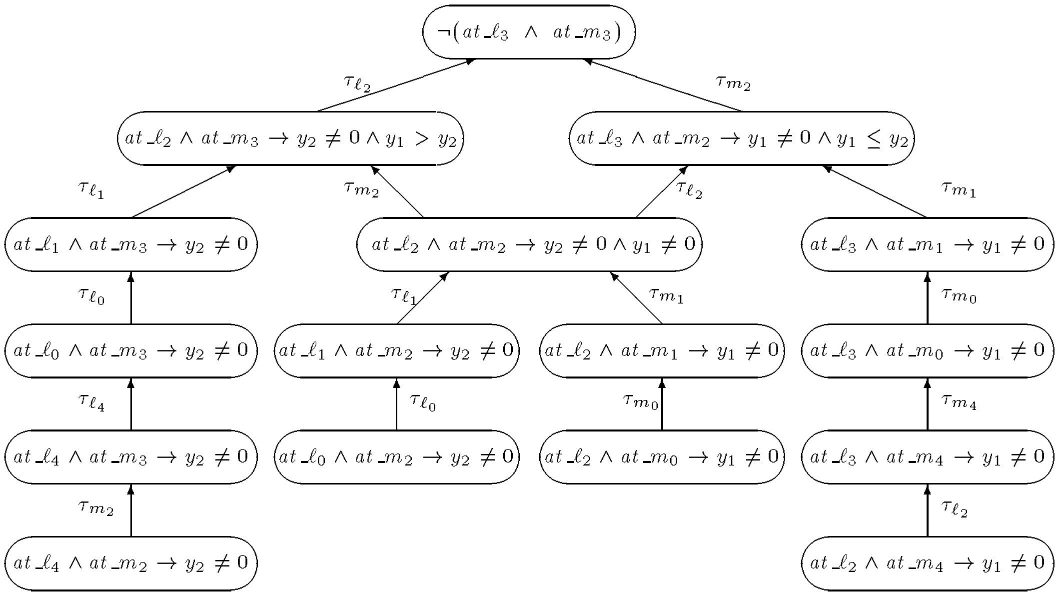


Fig. 5. Backward propagation from $\neg(at\_\ell_3 \ \wedge \ at\_m_3)$

one obtains the assertion labeling the source of the edge. The conjunction of the formulas is the greatest fixedpoint $\varphi_{\mathcal{B}}$ of $\mathcal{B}$. The auxiliary invariants $\square(y_1 \geq 0)$ and $\square(y_2 \geq 0)$ were used to simplify the examined conjuncts. These invariants can be generated automatically using for instance approximate forward propagation in the closed convex polyhedra domain, which is described later in this paper. Finally, since $\Theta: \ at\_\ell_0 \wedge at\_m_0 \wedge y_1 = y_2 = 0$ implies $\varphi_{\mathcal{B}}$, we have indeed established mutual exclusion of the critical sections.

The example shows the power of backward propagation: a completely automatic search terminates with a strengthened invariant. The analysis described above is entirely automatic in STeP. Forward propagation, on the other hand, does not converge in finitely many steps because the program's set of states cannot be reached in finitely many iterations.

4 General Safety

A *general safety* formula p is one equivalent to a temporal formula of the form

$$\square q$$

where q is a past formula, that is, a formula that does not contain any future temporal operators.

Example (general safety formulas) The state-causality formula

$$\Box\underbrace{[p \ \rightarrow \ \diamondminus r]}_{\text{past}}$$

for assertions p and r, is a general safety formula. A model satisfies this formula if every p-position coincides with or is preceded by an r-position.

The nested waiting-for formula

$$\Box[p \ \rightarrow \ q_1 \ \mathcal{W} \ (q_2 \ \mathcal{W} \ r)]$$

for assertions p, q_1, q_2 and r is a general safety formula. A model satisfies this formula if every p-position initiates a q_1-interval, followed by a q_2-interval, that may be terminated by an r-position. Each q_i-interval, $i = 1, 2$, is a set of successive positions all of which satisfy q_i, and may be empty or extend to infinity. In the latter case, there need not be a following interval nor a terminating r-position. This is a general safety formula, since it is equivalent to the formula

$$\Box\underbrace{[(\neg q_2) \ \rightarrow \ q_1 \ \mathcal{B} \ ((\neg p) \ \mathcal{B} \ r)]}_{\text{past}} \ .$$

■

4.1 Assertion Graphs

An *assertion graph* $G = \langle N, N_0, E, \lambda \rangle$ is a labeled directed graph where

N	–	set of nodes
$N_0 \subseteq N$	–	initial nodes
$E \subseteq N \times N$	–	edges
$\lambda : N \mapsto \mathcal{A}$	–	mapping from nodes to assertions.

G is an *assertion graph for* p if it describes exactly the models of p, i.e., for every model $\sigma : s_0, s_1, \ldots$, $\langle \sigma, 0 \rangle \vDash p$ iff there exists a path $n_0 (\in N_0), n_1, \ldots$ in G such that $s_i \vDash \lambda(n_i)$ for every $i \geq 0$.

Lemma 1 *Given a general safety formula p, where temporal operators do not appear in the scope of quantifiers, an assertion graph for p is computable.*

Proof (sketch)

To simplify the exposition we use the property that p is equivalent to a formula $\Box\varphi$ such that φ is a past temporal formula in which the only boolean operators are $\neg$ and $\wedge$, the only temporal operators are $\mathcal{B}$ and $\circleddash$ and no temporal operator is in the scope of a quantifier.

Let $cl(\varphi)$, the closure of φ, be the set of all subformulas f of φ and their negations. Thus, for example, $cl(p\ \mathcal{B}\ q) = \{p,\ q,\ \neg p,\ \neg q,\ p\ \mathcal{B}\ q,\ \neg(p\ \mathcal{B}\ q)\}$.

Let $G = \langle N, N_0, E, \lambda\rangle$ be the assertion graph with the following components

- $N \subseteq 2^{cl(\varphi)}$ where $n \in N$ iff
  - a) $\varphi \in n$,
  - b) for any $f \in cl(\varphi)$, $f \in n$ iff $\neg f \notin n$,
  - c) $f_1 \wedge f_2 \in n$ iff $f_1 \in n$ and $f_2 \in n$,
  - d) if $f_1\ \mathcal{B}\ f_2 \in n$ then $f_1 \in n$ or $f_2 \in n$.
- $N_0 \subseteq N$ where $n \in N_0$ iff $\circleddash f \notin n$ for every $\circleddash f \in cl(\varphi)$.
- $(n_1, n_2) \in E$ iff
  - a) for any $\circleddash f \in cl(\varphi)$, $\circleddash f \in n_2$ iff $f \in n_1$,
  - b) for any $f_1 \mathcal{B} f_2 \in cl(\varphi)$,
    $f_1 \mathcal{B} f_2 \in n_2$ iff either $f_2 \in n_2$ or $f_1\ \mathcal{B}\ f_2 \in n_1$.
- $\lambda(n) = \bigwedge\{f \in n \mid f \text{ is first-order}\}$.

It can be shown that G is an assertion graph for p. ▟

The outline given above is a simple modification of the well-known tableau construction [MP95]. One can also construct a tableau directly from the temporal formula without going into past-normal form. The construction requires us to consider more special cases, but guarantess that the assertion graph is of size $2^{O(|p|)}$.

Example (assertion graph) For program BAKERY in Figure 3, the 1-bounded overtaking property for process P_1 is expressed by the general safety formula p_{bound}:

$$\Box[at\_\ell_2 \ \rightarrow\ (at\_\ell_2 \wedge \neg at\_m_3)\ \mathcal{W}\ ((at\_\ell_2 \wedge at\_m_3)\ \mathcal{W}\ ((at\_\ell_2 \wedge \neg at\_m_3)\ \mathcal{W}\ at\_\ell_3))].$$

That is, whenever process P_1 reaches ℓ_2, process P_2 can access its critical region at most once before P_1 reaches its critical region at ℓ_3.

An assertion graph G for p_{bound} is shown in Figure 6. Statechart conventions [Har84] are used for a more compact graphical representation. For example, the edge departing the compound node containing n_1, n_2, n_3 to n_4 represents the edges $(n_1, n_4), (n_2, n_4), (n_3, n_4)$. The arrows without sources indicate initial nodes, thus, $N_0 = \{n_0, n_1, n_2\}$. The label at_ℓ_2 in the compound node distributes as a conjunct to the encapsulated nodes n_1, n_2 and n_3. ▟

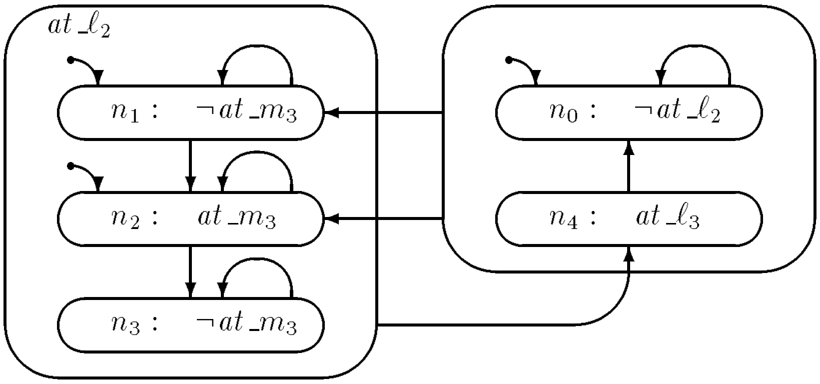


Fig. 6. An assertion graph G for property p_{bound}

For a node $n \in N$, we say that a state s is $(\mathcal{S}, n)$-*accessible* if there exists a finite computation $s_0, s_1, \ldots, s_k$, of $\mathcal{S}$, where $s = s_k$, and a finite path $n_0, n_1, \ldots, n_k$ in G such that $n_0 \in N_0$, $n = n_k$, and for $0 \leq i \leq k$, $s_i \vDash \lambda(n_i)$. An assertion φ is $(\mathcal{S}, n)$-*valid* if it holds for every $(\mathcal{S}, n)$-accessible state.

For transition system $\mathcal{S} = \langle \mathcal{V}, \Theta, \mathcal{T} \rangle$,
general safety formula p,
assertion graph $G = \langle N, N_0, E, \lambda \rangle$ for p, and
assertions $\{\varphi_n\}_{n \in N}$

$$\begin{array}{llll}
S1 & \Theta & \rightarrow \bigvee_{n \in N_0} \varphi_n & \\
S2 & \varphi_n & \rightarrow \lambda(n) & n \in N \\
S3 & \{\varphi_n\} \; \mathcal{T} & \{\bigvee_{(n,m) \in E} \varphi_m\} & n \in N \\
\hline
 & & \mathcal{S} \vDash p &
\end{array}$$

Fig. 7. Rule SAFE (general safety)

The verification rule SAFE shown in Figure 7 reduces the verification of the general safety formula p to first-order premises $S1$–$S3$. Suppose we have a transition system $\mathcal{S}$, a general safety formula p and an assertion graph G for p. To prove that p is $\mathcal{S}$-valid, we have to find *intermediate assertions* $\{\varphi_n\}_{n \in N}$ such that premise $S1$ is $\mathcal{S}$-state-valid and for each $n \in N$, $S2$ and $S3$ are $(\mathcal{S}, n)$-valid.

Lemma 2 *Rule* SAFE *is sound.*

Proof Assuming the premises of $S1$–$S3$ of SAFE are satisfied, we must show that p is $\mathcal{S}$-valid. Consider an arbitrary computation $\sigma : s_0, s_1, \ldots$ of $\mathcal{S}$, i.e.,

$$s_0 \vDash \Theta \quad \text{and} \quad \forall i \geq 0 \; \exists \tau \in \mathcal{T} \cdot (s_i, s_{i+1}) \vDash \rho_\tau(\overline{x}, \overline{x}') \ .$$

We show that $\sigma \vDash p$. By premise $S3$ we have

$$\forall i \geq 0\ \forall n \in N \cdot [s_i \vDash \varphi_n \quad \rightarrow \quad \exists m \cdot (n, m) \in E \ \wedge \ s_{i+1} \vDash \varphi_m] \ .$$

By premise $S1$, $\exists n \in N_0 \cdot s_0 \vDash \varphi_n$. Thus, by induction on i, it follows that $\forall i \geq 0\ \exists n \in N \cdot s_i \vDash \varphi_n$. Hence, by $S2$, σ induces a path $n_0(\in N_0), n_1, \ldots$ on G such that $s_i \vDash \lambda(n_i)$ for every $i \geq 0$. Therefore σ is a model of p. ▟

Lemma 3 *Rule* SAFE *is relatively complete for general safety formulas where temporal operators are not in the scope of quantifiers.*

Proof (sketch) Let p be an $\mathcal{S}$-valid general safety property with no temporal operators in the scope of a quantifier. By Lemma 1, there exists an assertion graph G for p. Assuming p is $\mathcal{S}$-valid we must establish that there are assertions $\{\varphi_n\}_{n \in N}$ satisfying premises $S1$–$S3$. By encoding finite sequences and corresponding operations in the assertion language $\mathcal{A}$ (assumed to be sufficiently expressive), it is possible to construct a formula $acc(\mathcal{S}, n)$ for each $n \in N$, such that:

$$s \vDash acc(\mathcal{S}, n) \quad \text{iff} \quad s \text{ is } (\mathcal{S}, n)\text{-accessible} \ .$$

The assertions $\{acc(\mathcal{S}, n)\}_{n \in N}$ trivially satisfy the premises of SAFE. ▟

Finding useful strengthened assertions $\{\varphi_n\}_{n \in N}$ for rule SAFE is the main obstacle in its use; assertions $\{acc(\mathcal{S}, n)\}_{n \in N}$ seldomly provide useful information for an automatic verifier. Similar to the analysis of INV we now give fixedpoint characterizations of the strongest and weakest intermediate assertion candidates for SAFE.

Notation: For an ordered set N we will denote a tuple, indexed by the elements in N, by $(a_n)_{n \in N}$ or a_N.

4.2 Forward Propagation

In general, rule SAFE, unlike rule INV, does not have a unique least set of intermediate assertions. Therefore we cannot hope to find, as we did in the invariance case, a forward operator whose least fixpoint gives the best possible intermediate assertions. The following is an example of an assertion graph that has two incomparable minimal (with respect to the order on $\mathcal{A}^{|N|}$ defined pointwise by the implication relation) tuples of assertions satisfying $S1$–$S3$.

Example (two minimal solutions for $S1{-}S3$) Consider a graph with two nodes, $N = N_0 = \{n_1, n_2\}$, labeled $\lambda(n_1) : x = 0$ and $\lambda(n_2) : y = 0$, in which any two nodes are connected. This graph is an assertion graph for the property $\square(x = 0 \vee y = 0)$. Consider the transition system with $\Theta : x = 0 \wedge y = 0$ and only one transition, τ_I, the idling transition, defined by $\rho_\tau : x' = x \wedge y' = y$. In this system there are two minimal solutions to $S1{-}S3$, namely

$$\begin{cases} \varphi_{n_1} : x = 0 \wedge y = 0 \\ \varphi_{n_2} : \mathrm{F} \end{cases} \qquad\qquad \begin{cases} \varphi_{n_1} : \mathrm{F} \\ \varphi_{n_2} : x = 0 \wedge y = 0 \ . \end{cases}$$

However, there is a subclass of graphs for which rule SAFE has a unique least set of intermediate assertions. This is the class of deterministic graphs. We say that an assertion graph G is *deterministic* if any model has at most one path in G starting in N_0, that is, for any n_1, n_2 that are different, if $n_1, n_2 \in N_0$ or for some $m \in N$, $(m, n_1), (m, n_2) \in E$ then $\lambda(n_1) \wedge \lambda(n_2)$ is unsatisfiable. Notice that in the above example G is not deterministic as $n_1, n_2 \in N_0$ and $\lambda(n_1) \wedge \lambda(n_2)$, which is $x = 0 \wedge y = 0$, is satisfiable.

We could restrict ourselves to deterministic graphs and still obtain a complete rule. But this is not necessary because we will show that we can deal with the general case by finding intermediate assertions for a stronger rule, STRONG-SAFE, shown in Figure 8, that admits a unique least set of intermediate assertions for any graph G.

For transition system $\mathcal{S} = \langle \mathcal{V}, \Theta, \mathcal{T} \rangle$,
general safety formula p,
assertion graph $G = \langle N, N_0, E, \lambda \rangle$ for p, and
assertions $\{\varphi_n\}_{n \in N}$

$$\begin{array}{llll} S1a & \Theta \ \wedge \ \lambda(n) & \rightarrow \varphi_n & n \in N_0 \\ S1b & \Theta & \rightarrow \bigvee_{n \in N_0} \lambda(n) & \\ S2 & \varphi_n & \rightarrow \lambda(n) & n \in N \\ S3a & \mathrm{SP}(\mathcal{T}, \varphi_n) \wedge \lambda(m) & \rightarrow \varphi_m & (n,m) \in E \\ S3b & \mathrm{SP}(\mathcal{T}, \varphi_n) & \rightarrow \bigvee_{(n,m) \in E} \lambda(m) & n \in N \\ \hline & & \mathcal{S} \models p & \end{array}$$

Fig. 8. Rule STRONG-SAFE (general safety modified)

Premises $S1a$ and $S1b$ strengthen premise $S1$ in SAFE, similarly $S3a$ and $S3b$ strengthen premise $S3$.

Lemma 4

(i) If a set of intermediate assertions satisfies the premises of rule STRONG-SAFE*, then it also satisfies the premises of rule* SAFE.

(ii) If G is deterministic then any set of intermediate assertions satisfying the premises of rule SAFE *also satisfies the premises of rule* STRONG-SAFE*.*

Proof 1. Let G be an assertion graph. Then $S1a$ and $S1b$ imply $S1$. It is also the case that $S3a$ and $S3b$ imply $S3$ since for $m \in N$:

$$\mathrm{SP}(\mathcal{T}, \varphi_n) \underbrace{\rightarrow}_{\text{by } S3b} \bigvee_{(n,m)\in E} \lambda(m) \rightarrow \bigvee_{(n,m)\in E} (\mathrm{SP}(\mathcal{T}, \varphi_n) \wedge \lambda(m)) \underbrace{\rightarrow}_{\text{by } S3a} \bigvee_{(n,m)\in E} \varphi_m \ .$$

2. Consider a deterministic assertion graph G. Then $S1$ and $S2$ imply $S1b$, and also $S1$ and $S2$ imply $S1a$ since

$$\begin{aligned} \Theta \wedge \lambda(n) &\underbrace{\rightarrow}_{\text{by } S1} \lambda(n) \wedge (\textstyle\bigvee_{n\in N_0} \varphi_n) \underbrace{\rightarrow}_{G \text{ det.}} \bigwedge_{\substack{m\neq n \\ m\in N_0}} \neg\lambda(m) \wedge \lambda(n) \wedge (\bigvee_{n\in N_0} \varphi_n) \\ &\underbrace{\rightarrow}_{\text{by } S2} \bigwedge_{m\neq n \wedge m\in N_0} \neg\varphi_m \wedge \lambda(n) \wedge (\bigvee_{n\in N_0} \varphi_n) \rightarrow \varphi_n \ . \end{aligned}$$

$S3$ implies $S3a$ since for $(n,m) \in E$:

$$\begin{aligned} \mathrm{SP}(\mathcal{T}, \varphi_n) \wedge \lambda(m) &\underbrace{\rightarrow}_{\text{by } S3} \lambda(m) \wedge (\bigvee_{(n,m')\in E} \varphi_{m'}) \\ &\underbrace{\rightarrow}_{G \text{ det.}} \lambda(m) \wedge \bigwedge_{\substack{(n,m')\in E \\ m'\neq m}} \neg\lambda(m') \wedge (\bigvee_{(n,m')\in E} \varphi_{m'}) \\ &\underbrace{\rightarrow}_{\text{by } S2} \lambda(m) \wedge \bigwedge_{\substack{(n,m')\in E \\ m'\neq m}} \neg\varphi_{m'} \wedge (\bigvee_{(n,m')\in E} \varphi_{m'}) \\ &\rightarrow \varphi(m) \ . \end{aligned}$$

$S2$ and $S3$ imply $S3b$, since $\mathrm{SP}(\mathcal{T}, \varphi_n) \underbrace{\rightarrow}_{\text{by } S3} \bigvee_{(n,m)\in E} \varphi_m \underbrace{\rightarrow}_{\text{by } S2} \bigvee_{(n,m)\in E} \lambda(m)$. ▟

Corollary 5 *Rule* STRONG-SAFE *is sound.*

Lemma 6 *Rule* STRONG-SAFE *is complete for general safety formulas where temporal operators are not in the scope of quantifiers.*

Proof The intermediate assertions given in the completeness proof for SAFE satisfy the premises of STRONG-SAFE. ▟

Rule STRONG-SAFE, unlike rule SAFE, has a unique least set of intermediate assertions (if any). We will show that whenever a set of intermediate assertions for rule STRONG-SAFE exists, a least such set is the fixedpoint of the following monotone operator $\mathcal{F}_N : \mathcal{A}^{|N|} \mapsto \mathcal{A}^{|N|}$ defined pointwise, at each $n \in N$, by

$$\mathcal{F}_n(\varphi_N) \quad \triangleq \quad \Big[(\Theta \wedge n \in N_0) \ \vee \ \bigvee_{(m,n)\in E} \mathrm{SP}(\mathcal{T}, \varphi_m)\Big] \ \wedge \ \lambda(n) \ .$$

Notice that a tuple of formulas φ_N satisfies $S1a$ and $S3a$ iff φ_N satisfies $\mathcal{F}_n(\varphi_N) \to \varphi_n$ for every $n \in N$.

If the sequence

$$(\mathrm{F})_{n\in N}, \quad \mathcal{F}_N((\mathrm{F})_{n\in N}), \quad \mathcal{F}_N^{(2)}((\mathrm{F})_{n\in N}), \quad \ldots$$

converges to $\varphi_{\mathcal{F},N}$ in finitely many steps, then $\varphi_{\mathcal{F},N}$ is the least fixedpoint of $\mathcal{F}_N$.

The connection between the rule STRONG-SAFE and $\varphi_{\mathcal{F},N}$ is given by the following lemma.

Lemma 7 *There exists a set of assertions satisfying the premises of rule* STRONG-SAFE *iff $S1b$ holds and $\varphi_{\mathcal{F},N}$ satisfies $S3b$. Thus, $\varphi_{\mathcal{F},N}$ precisely characterizes the $(\mathcal{S}, n)$-accesible states.*

Proof

$\Leftarrow$ Notice that $S2$ is always satisfied by $\varphi_{\mathcal{F},N}$, since $\varphi_{\mathcal{F},n} \leftrightarrow \mathcal{F}_n(\varphi_{\mathcal{F},N})$ and $\mathcal{F}_n(\varphi_{\mathcal{F},N}) \to \lambda(n)$ for $n \in N$. $\varphi_{\mathcal{F},N}$ is the least tuple $\varphi_{\mathcal{F},N}$ satisfying

$$\mathcal{F}_n(\varphi_{\mathcal{F},N}) \ \to \ \varphi_{\mathcal{F},n} \qquad \forall n \in N.$$

As we observed, this implies that $\varphi_{\mathcal{F},N}$ satisfies $S1a$ and $S3a$. If furthermore $S1b$ and $S3b$ hold, then $S1a - S3b$ are satisfied by $\varphi_{\mathcal{F},N}$.

$\Rightarrow$ Assume there is a tuple φ_N satisfying $S1a - S3b$. We have to show, that $S1b$ holds and $\varphi_{\mathcal{F},N}$ satisfies $S3b$. φ_N satisfies $S1a$ and $S3a$ and therefore

$$\mathcal{F}_n(\varphi_N) \ \to \ \varphi_n \qquad \forall n \in N.$$

Since $\varphi_{\mathcal{F},N}$ is the least fixedpoint of $\mathcal{F}_N$ we have

$$\varphi_{\mathcal{F},n} \ \to \ \varphi_n \qquad \forall n \in N.$$

Therefore, for every n,

$$\mathrm{SP}(\mathcal{T}, \varphi_{\mathcal{F},n}) \underbrace{\to}_{\mathrm{SP}\text{ monotonic}} \mathrm{SP}(\mathcal{T}, \varphi_n) \underbrace{\to}_{\text{by } S3b} \bigvee_{(n,m)\in E} \lambda(m)$$

hence $\varphi_{\mathcal{F},N}$ satisfies $S3b$. ▟

Corollary 8 *There exists a set of assertions satisfying the premises of rule* SAFE *if $S1b$ holds and $\varphi_{\mathcal{F},N}$ satisfies $S3b$.*

4.3 Extended Assertion Graphs

In order to show that a fixedpoint of the forward propagation operator is a good set of intermediate assertions we have to check that it satisfies condition $S3b$. If an iterative propagation method is used to find the fixedpoint, rather than checking $S3b$ after the fixedpoint has been reached, we can perform the check after each iteration. Checking $S3b$ at each iteration can save time and in case of failure help produce a counterexample. For a graphical interpretation of this method, we define the *(edge-complete) extension* G^e of G which is a supergraph of G with the property that any sequence whose first state satisfies $\bigvee_{n\in N_0^e}\lambda(n)$ has at least one path in G^e.

Suppose that n_f and $\{n^e\}_{n\in N}$ are new distinct symbols not in N. Let $G^e = \langle N^e, N_0^e, E^e, \lambda^e\rangle$, where

$$
\begin{array}{llll}
N^e = N \cup \{n_f\} \cup \{n^e \mid n \in N\} & \qquad & \lambda^e(n) & = \lambda(n) \quad \text{if } n \in N \\
N_0^e = N_0 & & \lambda^e(n^e) & = \neg(\bigvee_{(n,m)\in E}\lambda(m)) \\
E^e = E \cup \{(n,n^e),(n^e,n_f) \mid n \in N\} & & \lambda^e(n_f) & = \mathrm{T} \ .
\end{array}
$$

The n^e nodes can be considered as escape nodes for those computations that after reaching n have nowhere to go in G. All the computations that fail to stay in G end looping in the node n_f. Notice that if G is deterministic, using G^e we can characterize the models of both p and $\neg p$. A model σ satisfies p iff its (unique) path in G^e stays in N and conversely, it satisfies $\neg p$ iff its first state does not satisfy $\bigvee_{n\in N_0}\lambda(n)$ or its path in G^e reaches a node in $N^e \setminus N$.

Example (G **and** G^e) Consider program PROD-CONS (Figure 9), a simple version of a producer-consumer protocol. The statement **produce** x assigns some nonzero value to x and **consume** z sets z to 0. They do not change other variables. Statements **request** and **release** stand for the standard semaphores P and V. The causality property that a given value u is not consumed unless it was first produced can be expressed in temporal logic as

$$p_{caus}: \quad \Box\,[z = u \;\rightarrow\; \diamondminus\, x = u] \ .$$

The nodes in $N^e \setminus N$ in the assertion graph depicted in Figure 10 are drawn with two concentric ovals. ▟

$$\textbf{local}\ \ e, f\ \ : \textbf{integer where}\ e = 1, f = 0$$
$$x, y, z : \textbf{integer where}\ x = y = z = 0$$

$$Prod :: \left[\begin{array}{l}\textbf{loop forever do}\\ \quad\left[\begin{array}{ll}\ell_0: & \textbf{produce}\ x\\ \ell_1: & \textbf{request}\ e\\ \ell_2: & y := x\\ \ell_3: & \textbf{release}\ f\end{array}\right]\end{array}\right] \ \ || \ \ Cons :: \left[\begin{array}{l}\textbf{loop forever do}\\ \quad\left[\begin{array}{ll}m_0: & \textbf{request}\ f\\ m_1: & z := y\\ m_2: & \textbf{release}\ e\\ m_3: & \textbf{consume}\ z\end{array}\right]\end{array}\right]$$

Fig. 9. Program PROD-CONS

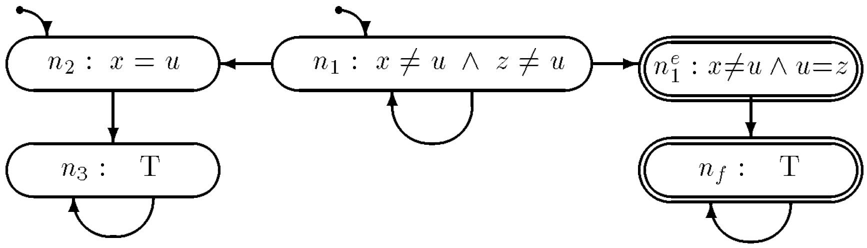


Fig. 10. An assertion graph G and its extension G^e for property p_{caus}

We extend the definition of the forward operator $\mathcal{F}_N$ to all the nodes of G^e, so for all $n \in N^e$,

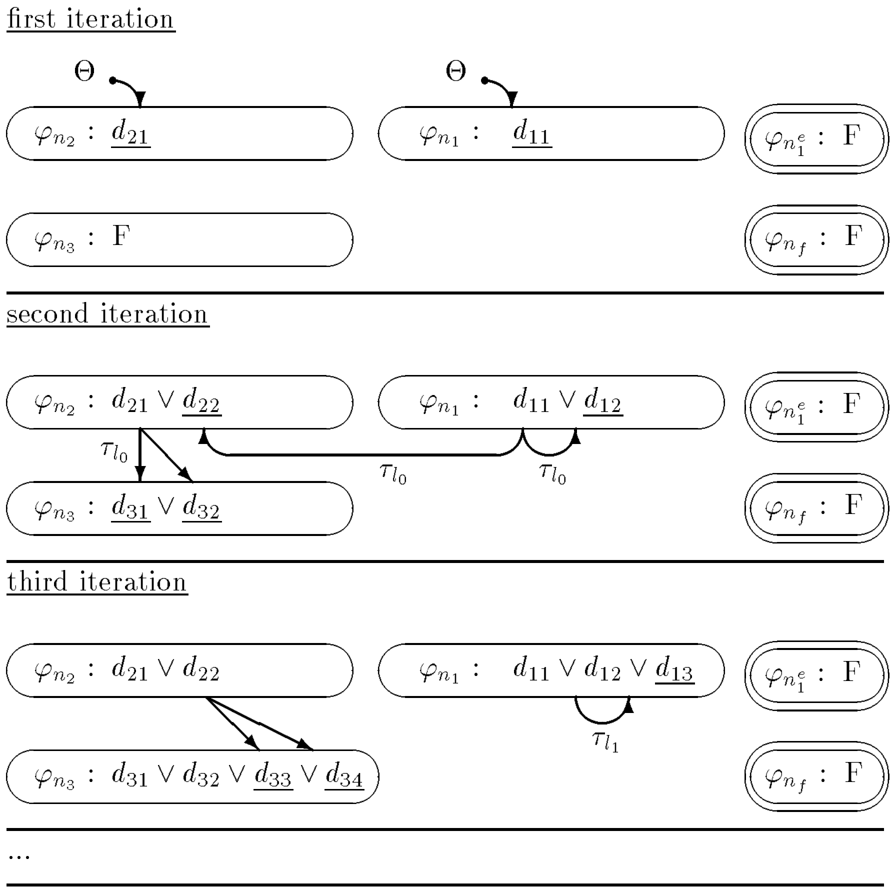
$$\mathcal{F}_n(\varphi_{N^e}) \ \ \triangleq\ \ \left[(\Theta \wedge n \in N_0^e) \vee \bigvee_{(m,n)\in E^e} \mathrm{SP}(\mathcal{T}, \varphi_m)\right] \ \wedge\ \lambda^e(n)\ .$$

With this definition, condition $S3b$ can be replaced by the condition that $\varphi_{\mathcal{F},n}$ is unsatisfiable for any $n \in N^e \setminus N$. If the sequence $(\mathrm{F})_{n\in N^e}$, $\mathcal{F}_{N^e}((\mathrm{F})_{n\in N^e})$, $\mathcal{F}_{N^e}^{(2)}((\mathrm{F})_{n\in N^e}), \ldots$ converges in finitely many steps to $\varphi_{\mathcal{F},N^e}$, then p is $\mathcal{S}$-valid iff $S1b$ holds and for any $n \in N^e \setminus N$, $\varphi_{\mathcal{F},n}$ is unsatisfiable.

Thus the bottom-up approach to verifying p consists of the following steps: check that $S1b$ holds, generate the least fixedpoint, $\varphi_{\mathcal{F},N^e}$, of $\mathcal{F}_{N^e}$ and check (at every step) that $\forall n \in N^e \setminus N$, $\varphi_{\mathcal{F},n}$ is unsatisfiable. Notice that, unlike the invariance case, forward propagation for general safety formulas depends on the given property.

Example (generating intermediate assertions for rule SAFE**)** Consider again program PROD-CONS (Figure 9), the temporal formula p_{caus}, and its assertion graph and extension in Figure 10.

The first three steps of the propagation sequence associated with G^e and program PROD-CONS are shown in Figure 11. Each new disjunct is underlined and its source is specified. For instance, disjunct d_{21} is obtained in the first iteration from the initial condition Θ. The disjunct d_{22} is obtained in the



where the disjuncts obtained are:

d_{11} :	ℓ_0	$\wedge$	m_0	$\wedge$	$x = 0$	$\wedge$	$(y,z) = (0,0)$	$\wedge$	$(e,f) = (1,0)$	$\wedge$	$u \neq 0$
d_{21} :	ℓ_0	$\wedge$	m_0	$\wedge$	$x = 0$	$\wedge$	$(y,z) = (0,0)$	$\wedge$	$(e,f) = (1,0)$	$\wedge$	$u = x$
d_{12} :	ℓ_1	$\wedge$	m_0	$\wedge$	$x \neq 0$	$\wedge$	$(y,z) = (0,0)$	$\wedge$	$(e,f) = (1,0)$	$\wedge$	$u \notin \{0,x\}$
d_{22} :	ℓ_1	$\wedge$	m_0	$\wedge$	$x \neq 0$	$\wedge$	$(y,z) = (0,0)$	$\wedge$	$(e,f) = (1,0)$	$\wedge$	$u = x$
d_{31} :	ℓ_0	$\wedge$	m_0	$\wedge$	$x \neq 0$	$\wedge$	$(y,z) = (0,0)$	$\wedge$	$(e,f) = (1,0)$	$\wedge$	$u = 0$
d_{32} :	ℓ_0	$\wedge$	m_0	$\wedge$	$x = 0$	$\wedge$	$(y,z) = (0,0)$	$\wedge$	$(e,f) = (1,0)$	$\wedge$	$u = x$
d_{13} :	ℓ_2	$\wedge$	m_0	$\wedge$	$x \neq 0$	$\wedge$	$(y,z) = (0,0)$	$\wedge$	$(e,f) = (0,0)$	$\wedge$	$u \notin \{0,x\}$
d_{33} :	ℓ_2	$\wedge$	m_0	$\wedge$	$x \neq 0$	$\wedge$	$(y,z) = (0,0)$	$\wedge$	$(e,f) = (0,0)$	$\wedge$	$u = x$
d_{34} :	ℓ_1	$\wedge$	m_0	$\wedge$	$x \neq 0$	$\wedge$	$(y,z) = (0,0)$	$\wedge$	$(e,f) = (1,0)$	$\wedge$	$u = x$

Fig. 11. First three iterations of the forward propagation process for program PROD-CONS and property p_{caus}

second iteration from the disjunct d_{21} via the transition τ_{l_0}. The propagation is completed by the ninth iteration with $\varphi_{n_e^1}$: F which proves that program PROD-CONS satisfies formula p_{caus}.

Notice that we do not have to compute $\varphi_{\mathcal{F},n_2}$ and $\varphi_{\mathcal{F},n_3}$. Since $N^e \setminus N$ is not reachable from these nodes it is not important what the actual formulas are. (We could take $\varphi_{\mathcal{F},n_2} : x = u$ and $\varphi_{\mathcal{F},n_3}$: T together with the other formulas to satisfy conditions $S1$–$S3$). ∎

The bottom-up method for proving a general safety formula p can also be used for debugging. With an assertion graph G and its extension G^e for p we generate assertions φ_{N^e}, expressing the $(\mathcal{S}, n)$-accessible states, in stages. Stage $i+1$ is obtained from stage i by applying $\mathcal{F}_{N^e}$. The propagation stops when we obtain a satisfiable $\varphi_{n,i}$ for $n \in N^e \setminus N$ and $i \geq 0$. By recording the history of previous iterations we can reconstruct the symbolic computation that ends outside G.

Example (counterexamples using forward propagation) Consider again program PROD-CONS (Figure 9). Suppose we want the protocol to have a lazy production property, namely that once the producer writes a value into x it will wait until that value is read by the consumer before assigning a new value to x. This property is expressed by the formula

$$p_{ord}: \quad \Box[x = u \ \rightarrow \ ((x = u \land z \neq u) \ \mathcal{W} \ (x = u \land z = u)) \ \ \mathcal{W} \ x \ \neq u]$$

where u is an auxiliary integer variable used to record the value of x. Figure 12 shows the graph G of all models of p_{ord}, and its extension G^e.

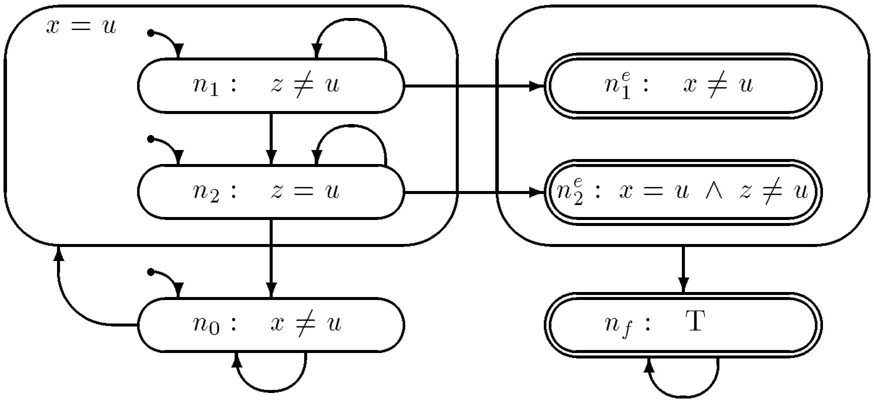


Fig. 12. Graphs G and G^e for the formula p_{ord}

Table 1 shows the disjuncts produced in the first six applications of the forward propagation operator $\mathcal{F}_N$ and maintains enough information to construct an abstract counterexample. Besides the conjunctions listed, each row contains the conjunct $z = 0$. We stopped when we obtained a satisfiable disjunct for a node in $N^e \setminus N$. A counterexample $s_1, s_2, \ldots, s_6, s_6, \ldots$ is generated by tracing back the origins of the first disjunct of $\varphi_{n_1^e}$:

$$\begin{array}{lll}
\text{in node } n_0: & s_1 & \models at_\ell_0 \land at_m_0 \land 0 = x \neq u \land y = 0 \land z = 0 \land e = 1 \land f = 0 \\
\text{in node } n_1: & s_2 & \models at_\ell_1 \land at_m_0 \land 0 \neq x = u \land y = 0 \land z = 0 \land e = 1 \land f = 0 \\
\text{in node } n_1: & s_3 & \models at_\ell_2 \land at_m_0 \land 0 \neq x = u \land y = 0 \land z = 0 \land e = 0 \land f = 0 \\
\text{in node } n_1: & s_4 & \models at_\ell_3 \land at_m_0 \land 0 \neq x = u \land y = u \land z = 0 \land e = 0 \land f = 0 \\
\text{in node } n_1: & s_5 & \models at_\ell_0 \land at_m_0 \land 0 \neq x = u \land y = u \land z = 0 \land e = 0 \land f = 1 \\
\text{in node } n_1^e: & s_6 & \models at_\ell_1 \land at_m_0 \land 0 \neq x \neq u \land y = u \land z = 0 \land e = 0 \land f = 1
\end{array}$$

	d	it.	src	disjunct
φ_{n_0}	d_{01}	1	Θ	$at\_\ell_0 \wedge at\_m_0 \wedge 0 = x \neq u \wedge y = 0 \wedge (e,f) = (1,0)$
	d_{02}	2	d_{01}	$at\_\ell_1 \wedge at\_m_0 \wedge 0 \neq x \neq u \wedge y = 0 \wedge (e,f) = (1,0)$
	d_{03}	3	d_{02}	$at\_\ell_2 \wedge at\_m_0 \wedge 0 \neq x \neq u \wedge y = 0 \wedge (e,f) = (0,0)$
	d_{04}	4	d_{03}	$at\_\ell_3 \wedge at\_m_0 \wedge 0 \neq x \neq u \wedge y = x \wedge (e,f) = (0,0)$
	d_{05}	5	d_{04}	$at\_\ell_0 \wedge at\_m_0 \wedge 0 \neq x \neq u \wedge y = x \wedge (e,f) = (0,1)$
	d_{06}	6	d_{05}	$at\_\ell_1 \wedge at\_m_0 \wedge 0 \neq x \neq u \wedge y \neq u \wedge (e,f) = (0,1)$
	d_{07}	6	d_{05}	$at\_\ell_0 \wedge at\_m_1 \wedge 0 \neq x \neq u \wedge y = x \wedge (e,f) = (0,0)$
φ_{n_1}	d_{11}	2	d_{01}	$at\_\ell_1 \wedge at\_m_0 \wedge 0 \neq x = u \wedge y = 0 \wedge (e,f) = (1,0)$
	d_{12}	3	d_{11}	$at\_\ell_2 \wedge at\_m_0 \wedge 0 \neq x = u \wedge y = 0 \wedge (e,f) = (0,0)$
	d_{13}	4	d_{12}	$at\_\ell_3 \wedge at\_m_0 \wedge 0 \neq x = u \wedge y = u \wedge (e,f) = (0,0)$
	d_{14}	5	d_{13}	$at\_\ell_0 \wedge at\_m_0 \wedge 0 \neq x = u \wedge y = u \wedge (e,f) = (0,1)$
	d_{15}	6	d_{14}	$at\_\ell_0 \wedge at\_m_1 \wedge 0 \neq x = u \wedge y = u \wedge (e,f) = (0,0)$
	d_{16}	6	d_{14}	$at\_\ell_1 \wedge at\_m_0 \wedge 0 \neq x = u \wedge y = u \wedge (e,f) = (0,1)$
	d_{17}	6	d_{05}	$at\_\ell_1 \wedge at\_m_0 \wedge 0 \neq x = u \wedge y \neq u \wedge (e,f) = (0,1)$
φ_{n_2}	d_{21}	1	Θ	$at\_\ell_0 \wedge at\_m_0 \wedge 0 = x = u \wedge y = 0 \wedge (e,f) = (1,0)$
$\varphi_{n_1^e}$	d_{31}	6	d_{14}	$at\_\ell_1 \wedge at\_m_0 \wedge 0 \neq x \neq u \wedge y = u \wedge (e,f) = (0,1)$
$\varphi_{n_2^e}$				

Table 1

Six iterations of forward propagation for PROD-CONS and p_{ord}.

The counterexample shows that it is possible that *Prod* produces a new value before *Cons* has had a chance to read the previous one. A fix to this problem is to switch statements ℓ_0 and ℓ_1. ▟

4.4 Backward Propagation

The situation is simpler for backward propagation, where a largest solution to $S2$ and $S3$ exists. The operator $\mathcal{B}_N : \mathcal{A}^{|N|} \mapsto \mathcal{A}^{|N|}$, on an assertion graph G, is defined pointwise as

$$\mathcal{B}_n(\varphi_N) \quad \triangleq \quad \lambda(n) \;\wedge\; \mathrm{WP}(\mathcal{T}, \textstyle\bigvee_{(n,m)\in E} \varphi_m) \enspace .$$

A tuple of formulas φ_N satisfies $S2$ and $S3$ iff it satisfies $\varphi_n \rightarrow \mathcal{B}_n(\varphi_N)$ for any $n \in N$. If the sequence $(\mathrm{T})_{n\in N}, \mathcal{B}_N((\mathrm{T})_{n\in N}), \mathcal{B}_N^{(2)}((\mathrm{T})_{n\in N}), \ldots$ converges in finitely many steps to a tuple of formulas $\varphi_{\mathcal{B},N}$ then $\varphi_{\mathcal{B},N}$ is the maximal tuple satisfying $S2$ and $S3$. The formula $\varphi_{\mathcal{B},n}$ characterizes those states s with the property that for any sequence of states $\sigma : s_0{=}s, s_1, \ldots$ originating in s such that $\forall i \geq 0 \cdot s_{i+1} \in \tau(s_i)$ there is a path $n_0{=}n, n_1, \ldots$ in G starting in n such that for any $i \geq 0$, $s_i \vDash \lambda(n_i)$. Furthermore,

$$p \text{ is } \mathcal{S}\text{-valid} \quad \text{iff} \quad \Theta \rightarrow \bigvee_{n\in N_0} \varphi_{\mathcal{B},n} \enspace .$$

Thus a top-down approach to verifying p consists of first generating the greatest fixedpoint, $\varphi_{\mathcal{B},N}$, of $\mathcal{B}_N$ and then checking that $\Theta \rightarrow \bigvee_{n\in N_0} \varphi_{\mathcal{B},n}$ holds.

Example (generating intermediate assertions for rule SAFE) Consider program BAKERY (Figure 3) and the 1-bounded overtaking property p_{bound}

$$\Box[at\_\ell_2 \;\rightarrow\; (at\_\ell_2 \wedge \neg at\_m_3)\; W\; ((at\_\ell_2 \wedge at\_m_3)\; W\; ((at\_\ell_2 \wedge \neg at\_m_3)\; W\; at\_\ell_3))]\;\; .$$

An assertion graph G for p_{bound} is shown in Figure 6. Forward propagation does not terminate, but the property can be proved using backward propagation. To compute the limit of the sequence $(\mathrm{T})_{n\in N}$, $\mathcal{B}_N((\mathrm{T})_{n\in N})$, $\mathcal{B}_N^{(2)}((\mathrm{T})_{n\in N}), \ldots$ we assume the invariant $\Box(at\_\ell_2 \rightarrow y_1 \geq 1)$.

This invariant can be generated by several known approximation methods [BBM95].

One iteration of $\mathcal{B}$ generates $\varphi_n = \lambda(n)$ for each $n \in N$. In the second iteration we get

$$\begin{aligned}
\varphi_{n_3} &= \lambda(n_3) \;\wedge\; \mathrm{WP}(\mathcal{T}, \lambda(n_3) \vee \lambda(n_4)) \\
&= (\neg at_m_3 \wedge at_\ell_2) \wedge \mathrm{WP}(\mathcal{T}, at_\ell_3 \vee (\neg at_m_3 \wedge at_\ell_2)) \\
&= (\neg at_m_3 \wedge at_\ell_2) \wedge \underbrace{(at_m_2 \rightarrow y_2 \geq y_1 \wedge y_1 \neq 0)}_{\mathrm{WP}(\tau_{m_2}, \neg at_m_3 \wedge at_\ell_2)} \\
&= (\neg at_m_3 \wedge at_\ell_2) \wedge \underbrace{(at_m_2 \rightarrow y_2 \geq y_1)}_{\text{by } \Box(at_\ell_2 \rightarrow y_1 \geq 1)}\;\; .
\end{aligned}$$

The other φ's remain at their previous values. A third application of $\mathcal{B}$ shows that a fixedpoint was reached. Since Θ implies φ_{n_0} and $n_0 \in N_0$, premise $S1$ holds. Backward propagation has thus provided

$$\begin{aligned}
&\varphi_{n_0} : \neg at_\ell_2, \\
&\varphi_{n_1} : \neg at_m_3 \wedge at_\ell_2, \\
&\varphi_{n_2} : at_m_3 \wedge at_\ell_2, \\
&\varphi_{n_3} : \neg at_m_3 \wedge at_\ell_2 \wedge (at_m_2 \rightarrow y_2 \geq y_1), \\
&\varphi_{n_4} : at_\ell_3
\end{aligned}$$

which are the necessary intermediate assertions to establish the 1-bounded overtaking property. ▟

5 Approximate Analysis

As we have seen, there is no guarantee for success in forward and backward analysis. For infinite or even large finite-state systems, the propagation may not terminate or, even if the sequences converge in a finite number of iterations, we may not be able to prove it (we might not be able to prove for instance that $\mathcal{F}^{n+1}(\mathrm{F}) \rightarrow \mathcal{F}^{n}(\mathrm{F})$). A solution to the difficulties in generating invariants is to limit the search for valid properties and intermediate assertions to a domain $\mathcal{D}$ where we can find approximations to the least (greatest) fixedpoint of the forward (backward) operator. At a minimum, the domain $\mathcal{D}$ is required to support a partial order $\leq$ corresponding to implication in the assertion language $\mathcal{A}$. The connection between $\mathcal{D}$ and $\mathcal{A}$ is given by a monotone function

$$\gamma : \mathcal{D} \mapsto \mathcal{A}$$

such that $d_1 \leq d_2$ implies $\gamma(d_1) \rightarrow \gamma(d_2)$. We say that $\mathcal{D}$ *abstracts* $\mathcal{A}$ and that γ is the *concretization function*. Abstract interpretation, introduced in [CC77], offers a general framework for the study of abstractions that preserve certain properties. It has been used to lift the analysis of these properties to a domain where it is easier to carry out.

Propagation performed on $\mathcal{D}$ will be called *approximate* analysis as opposed to the *exact* analysis one can perform on $\mathcal{A}$.

5.1 Invariance

To approximate a forward propagation operator

$$\mathcal{F} : \mathcal{A} \mapsto \mathcal{A}$$

on the assertion domain, to an abstraction $\mathcal{D}$, we need a monotone operator [2]

$$\overline{\mathcal{F}} : \mathcal{D} \mapsto \mathcal{D} \qquad \text{satisfying} \qquad \mathcal{F} \circ \gamma \rightarrow \gamma \circ \overline{\mathcal{F}}.$$

This property guarantees that, if for $d \in \mathcal{D}$:

$$\overline{\mathcal{F}}(d) \;\leq\; d$$

then

$$\mathcal{F}(\gamma(d)) \;\rightarrow\; \gamma(d),$$

and therefore premises $(I1)$ and $(I3)$ of rule INV are satisfied. The implication in the conclusion is based on two implications:

$$\mathcal{F}(\gamma(d)) \quad \underbrace{\rightarrow}_{\text{by } \mathcal{F}\circ\gamma\rightarrow\gamma\circ\overline{\mathcal{F}}} \quad \gamma(\overline{\mathcal{F}}(d)) \quad \underbrace{\rightarrow}_{\text{by monotonicity of } \gamma} \quad \gamma(d).$$

[2] The implication should be read as pointwise. Thus, for each $d \in \mathcal{D} \cdot \mathcal{F}(\gamma(d)) \rightarrow \gamma(\overline{\mathcal{F}}(d))$.

Thus, the task of verifying $\Box p$ can be reduced to that of finding a $d \in \mathcal{D}$, such that

$$\overline{\mathcal{F}}(d) \ \leq \ d \ \text{ and } \ \gamma(d) \ \rightarrow \ p \quad (I2).$$

Similarly, a given backward propagator $\mathcal{B}$ can be approximated by

$$\underline{\mathcal{B}} : \mathcal{D} \mapsto \mathcal{D} \qquad \text{satisfying} \qquad \gamma \circ \underline{\mathcal{B}} \ \rightarrow \ \mathcal{B} \circ \gamma$$

since if one can find a $d \in \mathcal{D}$ such that

$$d \ \leq \ \underline{\mathcal{B}}(d) \ \text{ and } \ \Theta \ \rightarrow \ \gamma(d) \quad (I1),$$

then $\gamma(d)$ satisfies premises $I1$–$I3$, and therefore $\Box p$ is valid.

Approximations $\overline{\mathcal{F}}$ and $\underline{\mathcal{B}}$ are only useful if the abstraction domain supports efficient methods to find solutions for $\overline{\mathcal{F}}(d) \leq d$, respectively $d \leq \underline{\mathcal{B}}(d)$.

5.2 Constructing Approximation Operators

The construction of the approximate propagation operators depends on the abstraction domain $\mathcal{D}$. This section identifies a set of assumptions about available operations on $\mathcal{D}$ that allow a uniform construction of these operators.

Since $\mathcal{F}$ is constructed using disjunction and SP it will be useful to have similar operations in $\mathcal{D}$. Assume therefore, as in abstract interpretation, that $\mathcal{D}$ is an upper semilattice with a join operator $\sqcup$, which is weaker than its boolean counterpart, i.e., for $d_1, d_2 \in \mathcal{D}$, $\gamma(d_1) \vee \gamma(d_2) \ \rightarrow \gamma(d_1 \sqcup d_2)$, and there are monotone

$$\mathrm{SP}^{\mathcal{D}} : \mathcal{D} \mapsto \mathcal{D} \qquad \text{satisfying} \qquad \mathrm{SP} \circ \gamma \ \rightarrow \ \gamma \circ \mathrm{SP}^{\mathcal{D}}$$

$$\Theta^{\mathcal{D}} : \mathcal{D} \qquad \text{satisfying} \qquad \Theta \ \rightarrow \ \gamma(\Theta^{\mathcal{D}}) \ .$$

Then we can define the forward propagation operator by:

$$\overline{\mathcal{F}}(d) \quad \triangleq \quad \Theta^{\mathcal{D}} \ \sqcup \ \mathrm{SP}^{\mathcal{D}}(\mathcal{T}, d).$$

By construction it will satisfy

$$\mathcal{F} \circ \gamma \ \rightarrow \ \gamma \circ \overline{\mathcal{F}}.$$

Similarly if $\mathcal{D}$ is a lower semilattice, with a strengthening meet operator $\sqcap$, i.e., for $d_1, d_2 \in \mathcal{D}, \gamma(d_1 \sqcap d_2) \;\rightarrow\; \gamma(d_1) \;\wedge\; \gamma(d_2)$, and there are monotone

$$\textsc{wp}^{\mathcal{D}} : \mathcal{D} \mapsto \mathcal{D} \qquad \text{satisfying} \qquad \gamma \circ \textsc{wp}^{\mathcal{D}} \;\rightarrow\; \textsc{wp} \circ \gamma$$

$$p^{\mathcal{D}} : \mathcal{D} \qquad \text{satisfying} \qquad \gamma(p^{\mathcal{D}}) \;\rightarrow\; p.$$

Then

$$\underline{\mathcal{B}}(d) \;\;\triangleq\;\; p^{\mathcal{D}} \;\sqcap\; \textsc{wp}^{\mathcal{D}}(\mathcal{T}, d)$$

satisfies

$$\gamma \circ \underline{\mathcal{B}} \;\rightarrow\; \mathcal{B} \circ \gamma.$$

Alternatively one can, as in constraint programming, require $\sqcup$ and $\sqcap$ to generate normal forms in $\mathcal{D}$, and require $\mathcal{D}$ to support an incremental satisfiability check [JM94].

5.3 Finitary propagation - invariance

We now study an example where forward propagation is performed on a finite propositional subset of the assertion language. This gives a finite abstraction domain $\mathcal{D}$, where the relation $\leq$ can be reduced to a propositional decision problem. The operator $\textsc{sp}^{\mathcal{D}}$ will however be defined in terms of $\textsc{sp}$, and this will require a restriction of first-order reasoning.

The parameterized program shown in Figure 13 implements a simple protocol for ordering access to location ℓ_3 among N processes. It uses two arrays j and a originally initialized to respectively 1 and F. Each process $P[n]$ starts at location ℓ_0 and resides in the while-loop, until every process $P[m]$ with lower index has set $a[m]$ to T.

For each i between 1 and N we associate transitions corresponding to the locations $\ell_0[i], \ell_1[i], \ell_2[i]$, and $\ell_3[i]$. For example the transition associated with location $\ell_0[i]$ moves control to $\ell_1[i]$ if $j[i] < i$, otherwise it moves control to $\ell_3[i]$.

The following two temporal properties specify the system's safety requirements

Invariance I: $\quad \Box[\; m < n \;\rightarrow\; \neg(\ell_3[m] \;\;\wedge\;\; \ell_3[n])\;]$

Precedence P: $\; m_0 < n_0 \;\rightarrow\; (\neg\ell_3[m_0] \;\;\wedge\;\; \neg\ell_3[n_0]) \;\; W \;\; (\ell_3[m_0] \;\;\wedge\;\; \neg\ell_3[n_0])$.

$$
\begin{array}{lll}
\textbf{in} & N & : \textbf{integer where } N > 0 \\
\textbf{local} & j & : \textbf{array } [1..\mathrm{N}] \ \textbf{ of } [1..\mathrm{N}] \ \textbf{ where } j = 1 \\
 & a & : \textbf{array } [1..\mathrm{N}] \ \textbf{ of boolean where } a = \mathrm{F}
\end{array}
$$

$$
||_{i=1}^{N} P[i] :: \left[\begin{array}{l}
\ell_0\text{: } \textbf{while } j[i] < i \textbf{ do} \\
\quad \left[\begin{array}{l} \ell_1\text{: } \textbf{await } a[j[i]] \\ \ell_2\text{: } j[i] := j[i] + 1 \end{array}\right] \\
\ell_3\text{: } a[i] := \mathrm{T} \\
\ell_4\text{:}
\end{array}\right]
$$

Fig. 13. Program ORDER

The invariance property implies that two different processes do not reside simultaneously at the critical location ℓ_3. The precedence property implies that processes with lower indices get priority entering the location ℓ_3.

Forward and backward propagation directly on the assertion language $\mathcal{A}$ is inadequate to handle parameterized transition systems. The unbounded number of processes necessitates introduction of quantifiers over the range $[1..N]$. Since each propagation step only considers the advance of one process, a potentially unbounded number of propagation steps are necessary to cover the search space.

One way to analyze the program in a finitary way is to restrict the assertion language over which program states can be described. In a way specifically tailored for the example, we restrict atomic propositions to be in one of the following forms:

$$
a[m] \quad a[n] \quad j[m] < n \quad j[m] = n \quad m < n \quad m \leq n \quad \ell_0[m]..\ell_4[m] \ .
$$

An assertion φ in this restricted language $\mathcal{D}$ is a boolean combination of these atomic propositions. An assertion in $\mathcal{A}$ is obtained projecting $\varphi \in \mathcal{D}$ to $\gamma(\varphi) = \forall n, m : [1..N].\varphi$ in $\mathcal{A}$. As $\sqcup$ we take disjunction $\vee$, which satisfies $\gamma(\varphi) \vee \gamma(\psi) \ \rightarrow \ \gamma(\varphi \ \vee \ \psi)$ since $(\forall x.\varphi) \vee (\forall x.\psi) \ \rightarrow \ \forall x.(\varphi \ \vee \ \psi)$. Control can be represented in a number of ways. Here, the most economical representation is in terms of one array $\pi : [1..N] \mapsto [0..4]$. So $\ell_3[i]$ is shorthand for $\pi[i] = 3$, which states that process $P[i]$ resides at location ℓ_3.

The abstraction of the initial condition in $\mathcal{D}$, is

$$
\Theta^{\mathcal{D}} : \ell_0[m] \ \wedge \ \neg a[m] \ \wedge \ j[m] \leq n \ .
$$

Since initially all processes reside at location ℓ_0, only transition $\ell_0[i]$ has any effect on the initial condition. The strongest post-condition $\mathrm{SP}(\mathcal{T}, \Theta^{\mathcal{D}})$ is therefore:

$$\exists \pi^p, i \cdot \left[\begin{array}{l} \ell_0^p[m] \;\wedge\; \neg a[m] \;\wedge\; j[m] \leq n \;\wedge\; \ell_0^p[i] \;\wedge \\ \left(\begin{array}{l} \pi = update(\pi^p, 1, i) \;\wedge\; j[i] < i \\ \vee \\ \pi = update(\pi^p, 3, i) \;\wedge\; j[i] \geq i \end{array} \right) \end{array} \right]$$

which does not belong to $\mathcal{D}$. We used $update(\pi, v, i)$ as shorthand for $\lambda j.\mathit{if}\ i = j\ \mathit{then}\ v\ \mathit{else}\ \pi[j]$. To eliminate the existential quantifier one distinguishes between two cases $i = m$ and $i \neq m$ to obtain the simplified form $\mathrm{SP}^{\mathcal{D}}(\mathcal{T}, \Theta^{\mathcal{D}})$:

$$\begin{array}{ll} & \ell_0[m] \;\wedge\; \neg a[m] \;\wedge\; j[m] \leq n \\ \vee & \ell_3[m] \;\wedge\; \neg a[m] \;\wedge\; j[m] \leq n \;\wedge\; j[m] = m \\ \vee & \ell_1[m] \;\wedge\; \neg a[m] \;\wedge\; j[m] \leq n \;\wedge\; j[m] < m \;\; . \end{array}$$

The subformula $j[m] = m$ is shorthand for $m = n \;\rightarrow j[m] = n$.

More formally we define $\mathrm{SP}^{\mathcal{D}}$ as $\alpha \;\circ\; \mathrm{SP} \;\circ\; \gamma$, where $\alpha : \mathcal{A} \mapsto \mathcal{D}$ is an abstraction mapping satisfying the *Galois insertion* properties: $\alpha \circ \gamma = Id$ and $\gamma \circ \alpha \leq Id$, that is for every $d \in \mathcal{D}$, $\alpha(\gamma(d)) = d$, and for every $\varphi \in \mathcal{A}$, $\gamma(\alpha(\varphi)) \rightarrow \varphi$. For this purpose it suffices to take $\alpha(\varphi)$ as the strongest possible $d \in \mathcal{D}$, such that $\models \varphi \rightarrow \gamma(d)$. Since validity ($\models$) in general is undecidable in $\mathcal{A}$ one can fix a maximal derivation length k and search for the strongest $d \in \mathcal{D}$ ($\mathcal{D}$ is in this example finite, and $\leq$ in $\mathcal{D}$ can be decided by propositional reasoning) such that $\vdash \varphi \rightarrow \gamma(d)$ with a derivation length at most k. Notice that $\mathrm{SP} \;\circ\; \gamma \rightarrow \;\gamma\; \circ\; \mathrm{SP}^{\mathcal{D}}$ as required.

The bottom-up search is presented in Figure 14. Each level corresponds to an iteration of $\overline{\mathcal{F}}$. The arrows indicate how $\overline{\mathcal{F}}$ follows the flow of control. Forward propagation $\overline{\mathcal{F}}$ itself works on the disjunction of all nodes below each level when obtaining the next level. The disjunction of the assertions listed in the graph's nodes is a fixedpoint of $\overline{\mathcal{F}}$. We refer to the fixedpoint as $\varphi_{\overline{\mathcal{F}}}$. The assertion $\forall m, n : [1..N].\varphi_{\overline{\mathcal{F}}}$ strengthens I and can be used successfully in rule INV to prove I.

In [CP93] unskolemization is applied to generate first-order intermediary assertions for proving partial correctness of flow-chart programs. Their approach resembles forward propagation by breadth-first search through candidate invariance properties. The candidates are generated using unskolemization as a weakening heuristic to obtain appropriate logical consequences.

The precedence property P is treated in Section 5.7.

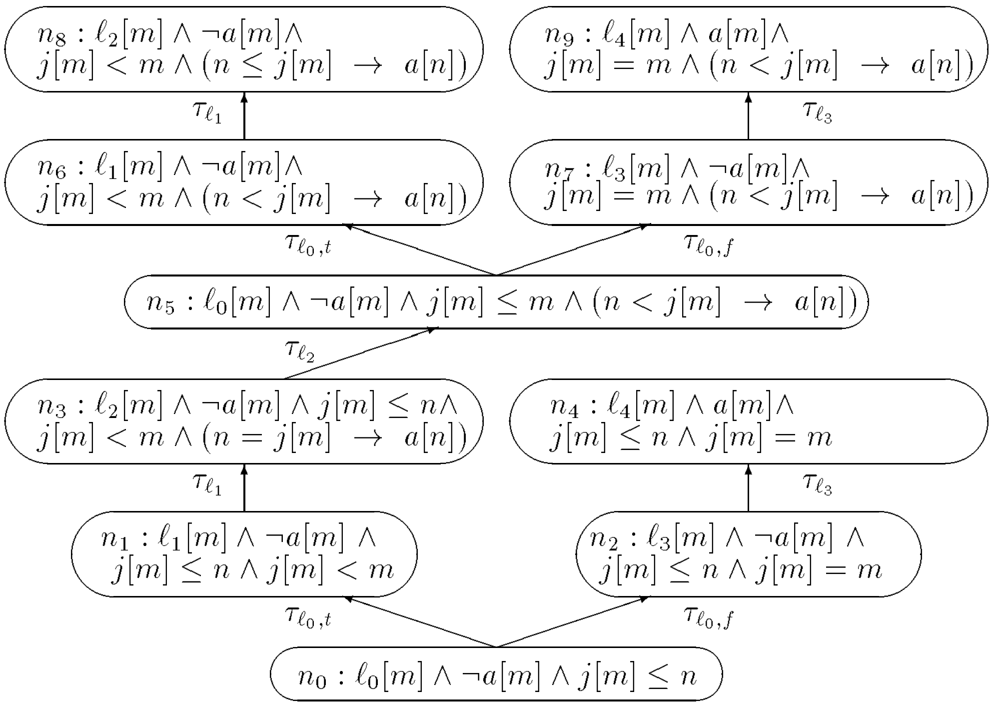


Fig. 14. Approximated state space for ORDER

5.4 Generating invariants in the closed convex polyhedra domain

The convex polyhedra abstraction was first studied as a tool in the analysis of programs in [CH78]. More recently, convex polyhedra have also been used in the analysis of linear hybrid systems [HRP94,HH95]. We first summarize the method (following [CH78]) and give an example of its use in the generation of invariants.

A closed convex polyhedron P is a formula $\bigwedge_i L_i$ where each L_i is a weak inequality, i.e.,

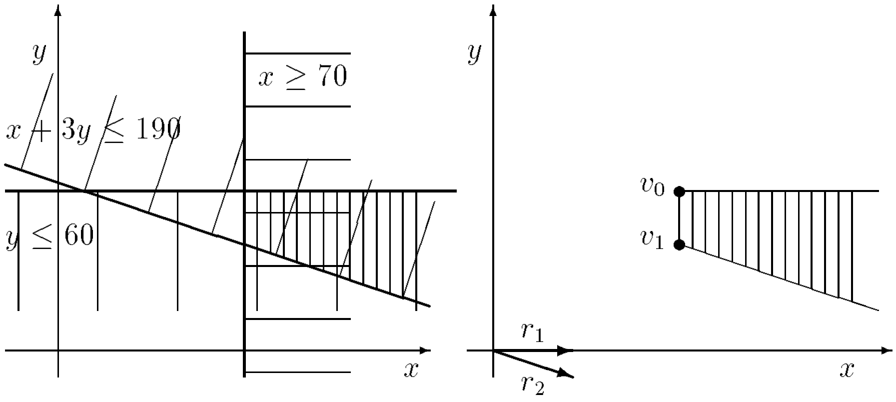
$$L_i = \sum_j a_{ij} x_j \geq b_i \ .$$

P can be viewed as the set of solutions of its system of linear constraints

$$P = \{\overline{x} \mid A\overline{x} \geq \overline{b}\} \ .$$

We say that $P = \{\overline{x} \mid A\overline{x} \leq \overline{b}\}$ *satisfies a linear constraint* $\overline{c} \cdot \overline{x} \leq d$ if $P = \{\overline{x} \mid A\overline{x} \leq \overline{b}, \overline{c} \cdot \overline{x} \leq d\}$, that is, $\overline{c} \cdot \overline{x} \leq d$ is linearly dependent on the constraints in $A\overline{x} \leq \overline{b}$.

Another representation of P is given by a system of generators: a finite set of



(a) Constraint representation (b) Generator representation
Fig. 15. Representations of a polyhedron

rays R and a finite set of *vertices* V such that

$$P = \left\{ \sum_{v_i \in V} \lambda_i \cdot v_i + \sum_{r_j \in R} \mu_j \cdot r_j \;\middle|\; \lambda_i \geq 0,\ \mu_j \geq 0 \text{ and } \sum_i \lambda_i = 1 \right\}$$

where P is the convex closure of this system of generators. Both representations are used in the standard inclusion decision procedure.

The domain Φ of closed convex polyhedra (with rational coefficients) is an abstraction of the assertion language $\mathcal{A}$ with the concretization function given by the inclusion $\gamma : \Phi \mapsto \mathcal{A}$. The partial order, minimal and maximal elements are the subset relation, the empty set, and the total set which are mapped by γ into the implication relation, F and T respectively.

An approximation of $\mathcal{F}$ in Φ is

$$\overline{\mathcal{F}}(\varphi) \quad \triangleq \quad \Theta^{\Phi} \sqcup \mathrm{SP}^{\Phi}(\mathcal{T}, \varphi), \text{ where :}$$

- $\sqcup$ is the *convex hull operator*: $P_1 \sqcup P_2$ is the smallest closed convex polyhedron that includes P_1 and P_2. The convex hull can be computed using the generator representations: if (R_1, V_1) and (R_2, V_2) are the generator representations for P_1 and P_2 then $(R_1 \cup R_2, V_1 \cup V_2)$ is a generator representation for $P_1 \sqcup P_2$.
- $\mathrm{SP}^{\Phi}(\mathcal{T}, \varphi) \triangleq \bigsqcup_{\tau} \mathrm{SP}^{\Phi}(\tau, \varphi)$, where $\mathrm{SP}^{\Phi}(\tau, \varphi)(V) \triangleq \exists V^p \cdot (\rho_{\tau}^{\Phi})(V^p, V) \wedge \varphi(V^p))$. Notice that $\exists x$ is an operator on Φ. Indeed, it can be shown that $\exists x.P = P \sqcup (\bigwedge_{y \neq x}(y = 0))$.
- $\varphi^{\Phi} \in \Phi$ is such that $\varphi \to \varphi^{\Phi}$. For a given φ we can compute such an φ^{Φ} by first rewriting $\varphi = c_1 \wedge \ldots \wedge c_k$ and then taking $\varphi^{\Phi} \triangleq \bigwedge\{c_i \mid c_i \text{ is linear}\}$.

The sequence

$$P_0 : \text{ F}, \quad P_1 : \overline{\mathcal{F}}(P_0), \quad P_2 : \overline{\mathcal{F}}(P_1), \ \ldots$$

does not necessarily converge in finitely many steps (it actually might not even converge in Φ). However it is possible to achieve convergence using a *widening operator* [CH78]. This is an operator $\bigtriangledown : \Phi \times \Phi \mapsto \Phi$ such that

(W**1**) for any $P_1, P_2 \in \Phi,\ P_1 \sqcup P_2 \rightarrow P_1 \bigtriangledown P_2$,
(W**2**) for any $P_0 \rightarrow P_1 \rightarrow \ldots$ in Φ, the sequence $P'_0,\ P'_1,\ \ldots$ defined by $P'_0 = P_0$ and $P'_{i+1} = P'_i \bigtriangledown P_{i+1}$ converges in a finite number of steps.

Several widening operators appear in the literature. The simplest is defined by

$$P_1 \bigtriangledown P_2 \quad \triangleq \quad \begin{cases} P_2 & \text{if } P_1 = \mathrm{F}, \\ \text{the polyhedron defined by the} & \\ \qquad \text{constraints of } P_1 \text{ satisfied by } P_2 & \text{otherwise} \end{cases}$$

The widening operator can be used to obtain the sequence

$$P'_0 :\ \mathrm{F}, \quad P'_1 :\ P'_0\ \bigtriangledown\ \overline{\mathcal{F}}(P'_0), \quad P'_2 :\ P'_1\ \bigtriangledown\ \overline{\mathcal{F}}(P'_1),\ \ldots.$$

which, by (W2), converges after finitely many iterations. The above sequence conservatively approximates $P_0, P_1, \ldots$ and therefore its limit is a solution of $\overline{\mathcal{F}}(P) \rightarrow P$.

Most widening operators have the property that the constraints of $P_1 \bigtriangledown P_2$ are a subset of the constraints of P_1 (if $P_1 \neq \mathrm{F}$). Thus the constraints of the limit polyhedron are a subset of the constraints of the polyhedron to which the first widening has been applied. For this reason, forward propagation with widening usually converges to a better solution if we start with a few iterations without widening. We will use this observation in our examples.

Example (generation of invariants) Consider program LOOP (Figure 16). We will use it to illustrate how we can obtain the basical loop invariants using

```
local   y  : integer where y = 0
        N  : integer where N ≥ 0

ℓ0: while y ≤ N − 1 do y := y + 1
ℓ1:
```

Fig. 16. Program LOOP

the polyhedra abstraction. Indeed, we will show how we can generate the invariant $0 \leq y \leq N$.

To illustrate the method and the need for a widening operator, we present in Figure 17 a few terms of the infinitely increasing sequence

$$P_1 : \overline{\mathcal{F}}(\mathrm{F}), \quad P_2 : \overline{\mathcal{F}}(P_1), \quad P_3 : \overline{\mathcal{F}}(P_2), \ldots$$

as well as a few terms of the sequence

$$P'_1 : \overline{\mathcal{F}}(\mathrm{F}), \quad P'_2 : P'_1 \sqcup \overline{\mathcal{F}}(P'_1), \quad P'_3 : P'_2 \bigtriangledown \overline{\mathcal{F}}(P'_2), \ldots$$

that converges in finitely many steps. A polyhedral forward propagation operator $\overline{\mathcal{F}}$ for LOOP can be defined from

$$\begin{array}{rcl}
\Theta^{\Phi} & \triangleq & y = 0 \wedge N \geq 0 \\
\rho^{\Phi}_{\tau_{\ell_0,T}} & \triangleq & y \leq N - 1 \wedge y' = y + 1 \\
\rho^{\Phi}_{\tau_{\ell_0,F}} & \triangleq & y \geq N \ .
\end{array}$$

For simplicity we ignore the two label variables ℓ_0 and ℓ_1. (Taking them into account generates 4-dimensional polyhedra.)

Partitioning the state space into sets that have the same location we could generate a stronger invariant for the transition system LOOP:

$$\Box\,[(\ell_0 \rightarrow 0 \leq y \leq N) \ \wedge \ (\ell_1 \rightarrow 0 \leq y = N)] \ . \ \blacksquare$$

5.5 *General Safety*

Approximations of the propagation operators follow the lines of simple invariance properties, only here the operators $\mathcal{F}_N$ and $\mathcal{B}_N$ work on N-tuples of assertions, and it will be convenient to extend operations γ and relations $\rightarrow$, $\leq$ pointwise to tuples.

Hence the forward operator

$$\mathcal{F}_N : \mathcal{A}^{|N|} \mapsto \mathcal{A}^{|N|}$$

defined on a graph G can be approximated by any weakening

$$\overline{\mathcal{F}}_N : \mathcal{D}^{|N|} \mapsto \mathcal{D}^{|N|} \quad \text{satisfying} \quad \mathcal{F}_N \circ \gamma \ \rightarrow \ \gamma \circ \overline{\mathcal{F}}_N.$$

This ensures, that whenever one can find a $d_N \in \mathcal{D}^{|N|}$, such that

$$\overline{\mathcal{F}}_N(d_N) \ \leq \ d_N$$

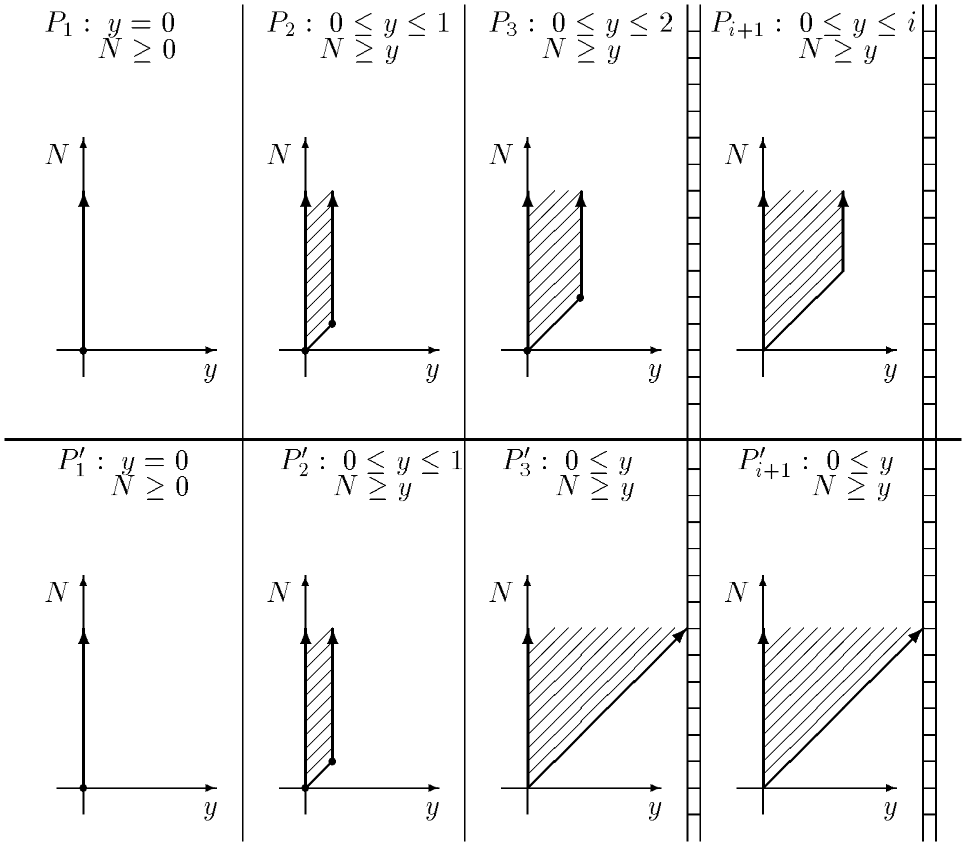


Fig. 17. The sequences $P_0, P_1, \ldots$ and $P'_0.P'_1, \ldots$

then

$$\mathcal{F}_N(\gamma(d_N)) \ \rightarrow\ \gamma(\overline{\mathcal{F}}_N(d_N)) \ \rightarrow\ \gamma(d_N) \ .$$

Any fixedpoint d_N of $\overline{\mathcal{F}}_N$ therefore corresponds to a tuple $\gamma(d_N)$, that satisfies $S1a$ and $S3a$. One can furthermore check if $\gamma(d_N)$ satisfies $S2$ and $S3b$, to see if it can be used to establish the safety property modeled by G. We notice, that if any fixedpoint of $\overline{\mathcal{F}}$ satisfies $S2$ and $S3b$, then the least fixedpoint will.

For backward propagation, the situation is analogous. The backward operator

$$\mathcal{B}_N : \mathcal{A}^{|N|} \mapsto \mathcal{A}^{|N|}$$

can be approximated by a strengthening

$$\underline{\mathcal{B}}_N : \mathcal{D}^{|N|} \mapsto \mathcal{D}^{|N|} \quad \text{satisfying} \quad \gamma \circ \underline{\mathcal{B}}_N \ \rightarrow\ \mathcal{B}_N \circ \gamma.$$

Then, whenever $d_N \in \mathcal{D}^{|N|}$ satisfies

$$d_N \ \leq\ \underline{\mathcal{B}}_N(d_N),$$

we have

$$\gamma(d_N) \;\rightarrow\; \gamma(\underline{\mathcal{B}}_N(d_N)) \;\rightarrow\; \mathcal{B}_N(\gamma(d_N)) \;.$$

Any fixedpoint of $\underline{\mathcal{B}}_N$ will therefore satisfy $S2$ and $S3$. The greatest fixedpoint will furthermore have the best chances to satisfy $S1$.

5.6 Constructing Approximation Operators

To obtain $\overline{\mathcal{F}}_N$ and $\underline{\mathcal{B}}_N$ uniformly from the labelings of the assertion graph $G = \langle N, N_0, E, \lambda\rangle$ we associate $\underline{G} = \langle N, N_0, E, \lambda^{\mathcal{D}}\rangle$, where the labeling function has been replaced by a stronger $\lambda^{\mathcal{D}}$, $\gamma \circ \lambda^{\mathcal{D}} \;\rightarrow\; \lambda$. For forward propagation we assume that $\mathcal{D}$, besides $\mathrm{SP}^{\mathcal{D}}$ and $\sqcup$, is equipped with a meet operator $\sqcap$, which is weaker than $\wedge$, i.e., $\gamma(d_1) \;\wedge\; \gamma(d_2) \;\rightarrow\; \gamma(d_1 \;\sqcap\; d_2)$ for every $d_1, d_2 \in \mathcal{D}$. We could take the definition of $\mathcal{F}_N$ and replace conjuntion with meet and disjunction with join. This results in

$$\overline{\mathcal{F}}_n(d_N) \triangleq \lambda^{\mathcal{D}}(n) \;\sqcap\; \left[\Theta^{\mathcal{D}} \wedge n \in N_0 \;\sqcup\; \mathrm{SP}^{\mathcal{D}}(\mathcal{T}, \sqcup_{(m,n)\in E}\, d_m)\right].$$

However, since in any lattice distributing $\sqcap$ inwards produces smaller results,

$$(c \sqcap a) \sqcup (c \sqcap b) \leq c \sqcap (a \sqcup b)$$

and since $\mathrm{SP}^{\mathcal{D}}$ is monotone, the definition

$$\overline{\mathcal{F}}_n(d_N) \triangleq (\lambda^{\mathcal{D}}(n) \sqcap \Theta^{\mathcal{D}} \wedge n \in N_0) \;\sqcup \bigsqcup_{(m,n)\in E} (\lambda^{\mathcal{D}}(n) \sqcap \mathrm{SP}^{\mathcal{D}}(\mathcal{T}, d_m))$$

results in a better approximation of $\mathcal{F}_N$. When $\mathrm{SP}^{\mathcal{D}}$ is expressed in terms of the individual transitions in $\mathcal{T}$, i.e., $\mathrm{SP}^{\mathcal{D}}(\mathcal{T}, d) = \sqcup_{\tau\in\mathcal{T}}\mathrm{SP}^{\mathcal{D}}(\tau, d)$, we distribute $\lambda^{\mathcal{D}}(n)$ further inwards. For brevity we therefore introduce the shorthand

$$\mathrm{SP}^{\mathcal{D}}_n(\tau, d) \;\triangleq\; \lambda^{\mathcal{D}}(n) \;\sqcap\; \mathrm{SP}^{\mathcal{D}}(\tau, d).$$

Analogously, for backward propagation we assume $\mathcal{D}$ equipped with a join operator $\sqcup$, which is stronger than disjunction, besides the stronger meet $\sqcap$. Then

$$\underline{\mathcal{B}}_n(d_N) \triangleq \lambda^{\mathcal{D}}(n) \;\sqcap\; \mathrm{WP}^{\mathcal{D}}(\mathcal{T}, \sqcup_{(n,m)\in E}\, d_m)$$

produces a useful approximation of $\mathcal{B}_N$.

Returning to the program in Figure 13 we can now establish the general safety property:

$$\text{Precedence} \quad m_0 < n_0 \ \rightarrow \ (\neg\ell_3[m_0] \ \wedge \ \neg\ell_3[n_0]) \ W \ (\ell_3[m_0] \ \wedge \ \neg\ell_3[n_0]) \ .$$

The systematic construction of $\overline{\mathcal{F}}_N$ gives a formal and automated argument for its validity. The assertion graph generated from the precedence property is given in Figure 18.

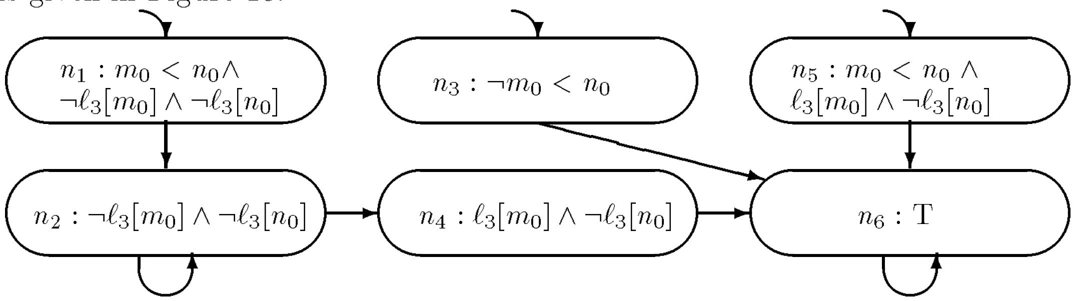


Fig. 18. Assertion graph for the Precedence property

The set of initial nodes is given by $N_0 = \{n_1, n_3, n_5\}$, and for $\overline{x} = (x_1, x_2, x_3, x_4, x_5, x_6)$ the assertion graph defines $\overline{\mathcal{F}}_N$ [3] by

$$\begin{array}{rcl}
\overline{\mathcal{F}}_{n_1}(\overline{x}) & = & \lambda(n_1) \ \wedge \ \Theta^{\mathcal{D}} \\
\overline{\mathcal{F}}_{n_2}(\overline{x}) & = & \lambda(n_2) \ \wedge \ \mathrm{SP}^{\mathcal{D}}(\mathcal{T}, x_1 \ \vee \ x_2) \\
\overline{\mathcal{F}}_{n_3}(\overline{x}) & = & \lambda(n_3) \ \wedge \ \Theta^{\mathcal{D}} \\
\overline{\mathcal{F}}_{n_4}(\overline{x}) & = & \lambda(n_4) \ \wedge \ \mathrm{SP}^{\mathcal{D}}(\mathcal{T}, x_2) \\
\overline{\mathcal{F}}_{n_5}(\overline{x}) & = & \lambda(n_5) \ \wedge \ \Theta^{\mathcal{D}} \\
\overline{\mathcal{F}}_{n_6}(\overline{x}) & = & \mathrm{T} \ .
\end{array}$$

We have thus extended the propositional language to contain atoms with the variables n_0 and m_0. The projection from $\mathcal{D}$ to the assertion language $\mathcal{A}$ is the same as before: $\varphi \in \mathcal{D}$ projects to $\gamma(\varphi) = \forall m, n : [1..N].\varphi$. As $\sqcap$ we have taken $\wedge$, which trivially satisfies the weakening condition, since $\gamma(\varphi \wedge \psi)$ is in fact equivalent to $\gamma(\varphi) \ \wedge \ \gamma(\psi)$. To find $\varphi_{\overline{\mathcal{F}},N} \in \mathcal{D}^{|N|}$ such that $\overline{\mathcal{F}}_N(\varphi_{\overline{\mathcal{F}},N}) \leq \varphi_{\overline{\mathcal{F}},N}$ it suffices to find a solution for x_2 in

$$x_2 \ \leq \ \neg\ell_3[m_0] \wedge \neg\ell_3[n_0] \wedge \mathrm{SP}^{\mathcal{D}}(\mathcal{T}, (m_0 < n_0 \wedge \neg\ell_3[m_0] \wedge \neg\ell_3[n_0] \wedge \Theta^{\mathcal{D}}) \ \vee \ x_2).$$

The formula $\varphi_{\overline{\mathcal{F}}} \wedge \ m_0 < n_0 \ \wedge \ \neg\ell_3[m_0] \ \wedge \ \neg\ell_3[n_0]$ works here, and is generated in the same way as $\varphi_{\overline{\mathcal{F}}}$ was in Figure 14. The rest of $\varphi_{\overline{\mathcal{F}},N}$ is extracted directly

[3] The equation for $\overline{\mathcal{F}}_{n_6}$ does not follow the general guidelines for construction of $\overline{\mathcal{F}}$, but is obtained by optimization based upon the label in n_6.

from the definition of $\overline{\mathcal{F}}_N$. Checking the resulting $\varphi_{\overline{\mathcal{F}},N}$ against $S3b$ finally establishes the validity of the precedence property.

5.8 General Safety Analysis in the closed convex polyhedra domain

Let G be an assertion graph and $\underline{G}$ a strengthened graph labeled by $\lambda^{\Phi} : N \mapsto \Phi \subseteq \mathcal{A}$ such that

$$\lambda^{\Phi}(n) \rightarrow \lambda(n) \ .$$

In the exact analysis we have used extension graphs to stop the propagation as soon as we have found a computation that cannot be contained in the assertion graph. It is possible to use this method in the approximate analysis as well, however $\lambda(n^e) = \neg(\bigvee_{(n^e,m)\in E} \lambda(m))$ has to be replaced by the weaker condition $\neg(\bigvee_{(n,m)\in E} \lambda^{\Phi}(m)) \rightarrow \lambda^{\Phi}(n^e)$. This condition ensures that if P_{N^e} are a solution of the forward propagation operator on $\underline{G}^e$ and all the polyhedra corresponding to nodes in $\underline{G}^e - \underline{G}$ are empty then the systems satisfies the specification.

In the analysis of non-parameterized systems we can define an approximation of $\mathcal{F}_{N^e}$ in Φ as in Section 5.6. For parameterized transitions, we will present later in this section one possible definition of $\mathrm{SP}_n^{\Phi}(\tau, P)$ and of the concretization function γ such that $\mathrm{SP}_n(\tau, \gamma(P)) \ \rightarrow \ \gamma(\mathrm{SP}_n^{\Phi}(\tau, P))$. Under these conditions, $\overline{\mathcal{F}}_{N^e}$ has all the desired properties, except that the ascending sequence

$$\underbrace{(\mathrm{F})_{n\in N^e}}_{P_{N^e,0}}, \ \ \underbrace{\overline{\mathcal{F}}_{N^e}(P_{N^e,0})}_{P_{N^e,1}}, \ \ \underbrace{\overline{\mathcal{F}}_{N^e}(P_{N^e,1})}_{P_{N^e,2}}, \ \ldots$$

does not necessarily converge. We could use again a widening operator to speed up the convergence process. However, the widening operator of Section 5.4 does not guarantee that a solution φ_{N^e} has the property that $\forall n \in N^e {\cdot} \varphi_n \rightarrow \lambda^{\Phi}(n)$. This motivates the following definition:

A *bounded widening operator* is an operator $\bigtriangledown : (\varphi_1, \varphi_2, \psi) \mapsto \varphi_1 \overset{\psi}{\bigtriangledown} \varphi_2$, where

(W1′) for any $P_1, P_2, P \in \Phi$, $P_1 \sqcup P_2 \rightarrow P_1 \overset{P}{\bigtriangledown} P_2$
(W2′) for any $P_0 \ \rightarrow \ P_1 \ \rightarrow \ \ldots$ in Φ and $P \in \Phi$, the sequence $P'_0, \ P'_1, \ \ldots$ defined by $P'_0 = P_0$ and $P'_{i+1} = P'_i \overset{P}{\bigtriangledown} P_{i+1}$ converges in finitely many steps.
(W3) for any $P_1, P_2, P \in \Phi$, if $P_1 \rightarrow P$ and $P_2 \rightarrow P$ then $P_1 \overset{P}{\bigtriangledown} P_2 \rightarrow P$.

An example of such a bounded operator is

$$P_1 \overset{P}{\triangledown} P_2 = \begin{cases} P_2 & \text{if } P_1 = \mathrm{F} \\ \text{the polyhedron defined by} & \\ \qquad \text{the constraints of } P_1 \text{ satisfied by } P_2 \text{ and} & \\ \qquad \text{the constraints of } P \text{ satisfied by } P_1, P_2 & \text{otherwise} \end{cases}$$

For a forward operator bounded by P the bounded widening operator can be used to obtain converging sequences bounded by P. For any $n \in N^e$, the sequence

$$P'_{n,0} : \ \mathrm{F}, \quad P'_{n,1} : \ P'_{n,0} \overset{\lambda^\Phi(n)}{\triangledown} \overline{\mathcal{F}}_n(P'_{N,0}), \quad \ldots$$

converges in finitely many steps to a limit that is bounded by $\lambda^\Phi(n)$.

Example The program MUX-AST presented in Figure 19 implements mutual exclusion by add-and-store instructions. The M processes coordinate their entry to the critical sections using the shared variable y. Each process has a local variable t. A non-negative value for t signals that the process can enter its critical section.

```
in     M : integer where M > 0
local  y : integer where y = 1

              [local t : integer
               ℓ0: : loop forever do
                     [ℓ1: noncritical
                      ℓ2: t := −1
                      ℓ3: while t < 0 do
||_{i=1}^M P[i] ::        [ℓ4: await y > 0
                           ℓ5: (t, y) := (y − 1, y − 1)
                           ℓ6: if t = 0 then
                                  ℓ7: critical
                           ℓ8: y := y + 1                ]]]
```

Fig. 19. Program MUX-AST

The property we want to verify is that if process k reaches ℓ_6 and no other process is at ℓ_6, ℓ_7 or ℓ_8 then no other process will access its critical region before process k. For this we will introduce the auxiliary variables N_i for $i \in \{6,7,8\}$ that will count how many processes are at location ℓ_i. An augmented transition system takes N_i into account:

$$\begin{array}{ll} \widehat{\Theta}: & \Theta \ \wedge \ N_6 = N_7 = N_8 = 0 \\ \widehat{\rho}_{\tau_{\ell_5}[i]}: & \rho_{\tau_{\ell_5}[i]} \ \wedge \ N_6' = N_6 + 1 \ \wedge \ N_7' = N_7 \ \wedge \ N_8' = N_8 \\ \ldots & \end{array}$$

It can be shown that any computation of the original transition system corresponds to a computation of the augmented system. This justifies its use in the verification process. The property can now be expressed in temporal logic as

$$\begin{array}{lll} p_{\mathit{first}}: & 1 \leq k \neq m \leq M \ \rightarrow & ((\ell_6[k] \ \wedge \ N_{6..8} = 1 \ \wedge \ \ominus \ell_5[k]) \ \Rightarrow \\ & & (\ell_6[k] \ \wedge \ \neg \ell_7[m]) \ \mathcal{W} \ \ \ell_7[k]) \end{array}$$

where $N_{6..8}$ is shorthand for $N_6 + N_7 + N_8$.

This property cannot be proven using forward propagation in Φ by itself. However, it is possible to prove it by combining forward propagation in Φ with other invariant generation methods. An established invariant $\square \chi$ can be used in the approximate analysis to strengthen the labelings of all the nodes in the assertion graph or an extended supergraph of the assertion graph. The invariant used in this example is

$$\square \chi \ = \ \square \forall j \in [1..M] \cdot (\chi_1[j] \ \wedge (1 \leq k \neq j \leq M \ \rightarrow \ \chi_2[j]))$$

where

$$\begin{array}{lll} \chi_1[j] & = & \ell_6[j] \leq N_6 \ \wedge \ \ell_7[j] \leq N_7 \ \wedge \ \ell_8[j] \leq N_8 \\ \chi_2[j] & = & \ell_6[k] + \ell_6[j] \leq N_6 \ . \end{array}$$

An assertion graph G_{first} for this property is shown in Figure 20. For ease of presentation, we have omitted some of the unreachable nodes. For instance we have eliminated the nodes that contained the conjunct $\ell_5[k] \ \wedge \ \ell_6[k]$. We have also omitted the nodes generated from node n_0 as no node in $\underline{G}^e_{\mathit{first}} - \underline{G}_{\mathit{first}}$ is reachable from it.

A node of an assertion graph for φ can be split into a cluster of nodes each corresponding to a disjunct of its labeling. The resulting graph is also an assertion graph for φ. Such a splitting results in a tighter approximation in the polyhedra domain and is therefore done whenever possible. Another method used in this example to obtain a better polyhedral approximation is to rewrite inequalities involving integer variables as a disjunction of weak inequalities. For example, the node n_4, labeled by $\lambda(n_4): k \neq m \ \wedge \ \ell_5[k]$ can split into two nodes labeled $m \leq k-1 \ \wedge \ \ell_5[k]$ and $m \geq k+1 \ \wedge \ \ell_5[k]$ respectively.

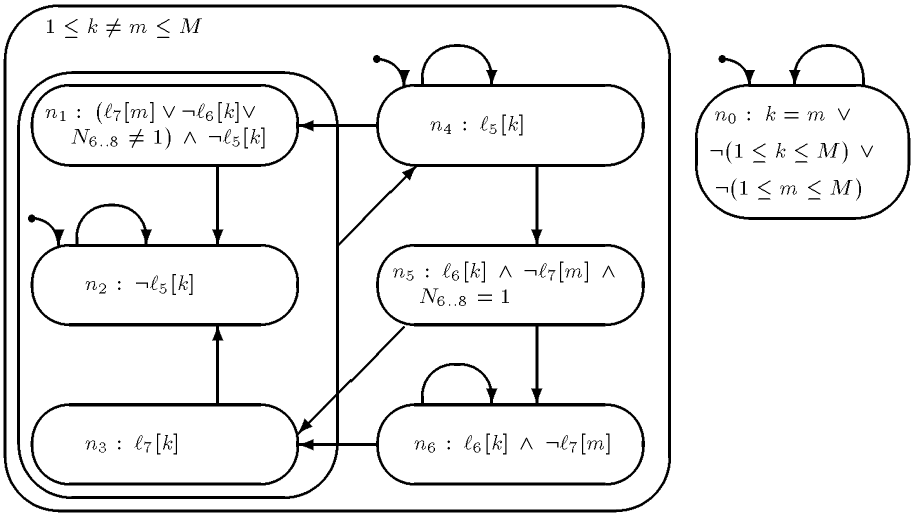


Fig. 20. The assertion graph G_{first} for the formula p_{first}

A strengthened graph $\underline{G}_{\mathit{first}}$ and an extension of it are presented in Figure 21. The variables considered in the analysis are the shared variables of the processes and the local variables of the processes k, m and of a generic process $i \neq k$. For a process $P[j]$, let V_j be the local variables $t, \ell_5, \ell_6, \ell_7$ corresponding to process $P[j]$. The set of variables considered in the analysis is then $\{M, N_6, N_7, N_8, y, k, m, i\} \cup V_k \cup V_m \cup V_i$. The labelings of all the nodes G_{first} except n_0 contain the conjunct $m \neq k$. Therefore all these nodes were split into copies corresponding to $m \leq k-1$ and $m \geq k+1$ respectively. For convenience, only the nodes for $m \leq k-1$ are represented in Figure 21. All the nodes in $\underline{G}^e_{\mathit{first}}$ are labeled either by $i \leq k-1$ or $i \geq k+1$ and again only the nodes for $i \leq k-1$ have been represented. The conjunct $\chi^\Phi = \chi_1[k] \wedge \chi_1[m] \wedge \chi_1[i] \wedge \chi_2[m] \wedge \chi_2[i]$ in the labelings of the nodes for which $k \neq m$ is an approximation of χ in Φ. Indeed $\chi \wedge m \neq k \ \rightarrow \ \gamma(\chi^\Phi)$.

The forward propagation operator associated with $\underline{G}^e_{\mathit{first}}$ is defined as in Section 5.6 with $\textsc{sp}^\Phi_n(\tau, \varphi)$ given by

$$\begin{array}{rcl}
\textsc{sp}^\Phi_n(\tau,\varphi) & \triangleq & \quad \textsc{sp}^\Phi_n(\tau[k],\varphi) \\
& & \sqcup \ \textsc{sp}^\Phi_n(\tau[m],\varphi) \\
& & \sqcup \ \textsc{sp}^\Phi_n(\tau[i],\varphi) \ \wedge \ 1 \leq i \leq k-1 \\
& & \sqcup \ \textsc{sp}^\Phi_n(\tau[i],\varphi) \ \wedge \ k+1 \leq i \leq M \\
& & \sqcup \ \exists i, V_i \cdot \left(\begin{array}{l} \quad \textsc{sp}^\Phi_n(\tau[i],\varphi) \ \wedge \ 1 \leq i \leq k-1 \\ \sqcup \ \textsc{sp}^\Phi_n(\tau[i],\varphi) \ \wedge \ k+1 \leq i \leq M \end{array} \right) .
\end{array}$$

As concretization function we take γ defined by

$$\gamma(P[i]) = \forall i \in [1..M] \cdot (i \neq k \rightarrow P[i]).$$

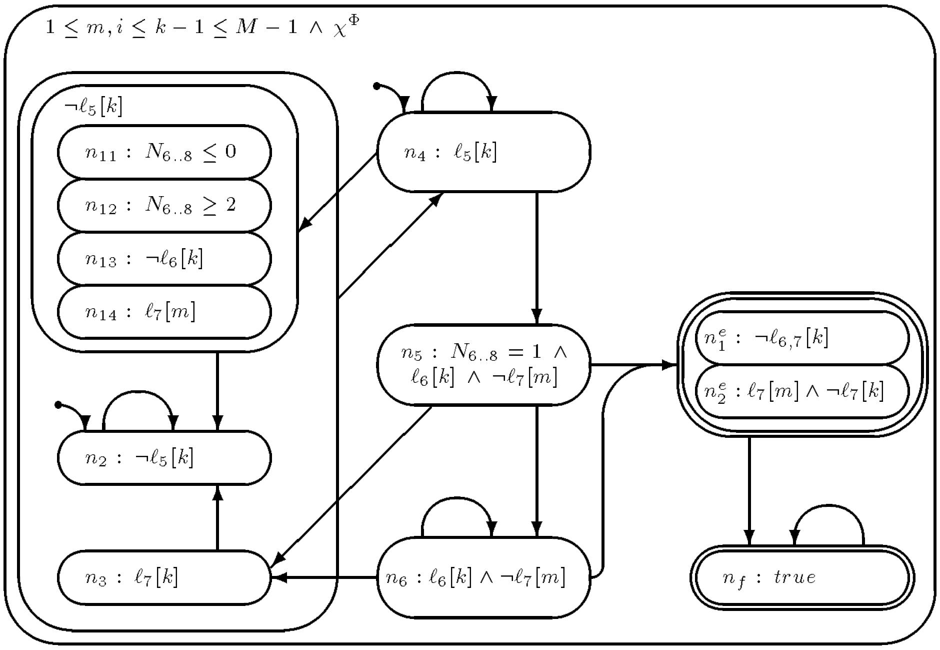


Fig. 21. A strengthened graph $\underline{G}_{\mathit{first}}$ for G_{first} and an extension $\underline{G}^e_{\mathit{first}}$ for it, where $\chi^{\Phi} = \chi_1[k] \wedge \chi_1[m] \wedge \chi_1[i] \wedge \chi_2[m] \wedge \chi_2[i]$.

It can be shown that $\mathrm{SP}_n(\{\tau[j]\}_j, \gamma(P[i])) \rightarrow \gamma(\mathrm{SP}^{\Phi}_n(\{\tau[j]\}_j, P[i]))$ and therefore this will define a good approximation of the forward propagation operator.

Forward propagation for the nodes in Figure 21 results in the following polyhedra

$$
\begin{array}{lcll}
P_s & = & \psi \ \wedge \ \lambda^{\Phi}(n_s) & \text{for } n_s \in \{n_{11}, n_{12}, n_{13}, n_{14}, n_2, n_4\} \\
P_3 & = & \psi \ \wedge \ \lambda^{\Phi}(n_3) \ \wedge \ t[k] = 0 \ \wedge \ N_7 = 1 \ \wedge \ t[m] + \ell_6[m] \leq 0 \ \wedge \ t[i] + \ell_6[i] \leq 0 & \\
P_5 & = & \psi \ \wedge \ \lambda^{\Phi}(n_5) \ \wedge \ t[k] = 0 & \\
P_6 & = & \psi \ \wedge \ \lambda^{\Phi}(n_6) \ \wedge \ t[k] = 0 \ \wedge \ N_7 = 0 \ \wedge \ t[m] + \ell_6[m] \leq 0 \ \wedge \ t[i] + \ell_6[i] \leq 0 & \\
P_1^e & = & P_2^e \ = \ P_f \ = \ \mathrm{F} &
\end{array}
$$

where

$$
\begin{array}{rcl}
\psi & = & \psi_{inv} \ \wedge \ \psi_l[m] \ \wedge \ \psi_l[k] \ \wedge \ \psi_l[i] \\
\psi_{inv} & = & m \leq k-1 \ \wedge \ i \leq k-1 \ \wedge \ N_{6..8} + y = 1 \\
\psi_l[j] & = & 1 \leq j \leq M \ \wedge \ \ell_{5..7}[j] \leq 1 \ \wedge \ t[j] \leq 0 \ \wedge \ \psi_{ls}(\{\ell_5[j], \ell_6[j], \ell_7[j]\}) \\
\psi_{ls}(vs) & = & \bigwedge_{v \in vs} 0 \leq v \leq 1 \ .
\end{array}
$$

The propagation on the nodes with $m \geq k+1$ or $i \geq k+1$ yields similar results. As all the polyhedra in $\underline{G}^e_{\mathit{first}} - \underline{G}_{\mathit{first}}$ are empty we have $\text{MUX-AST} \vDash p_{\mathit{first}}$. ▪

6 Conclusion

This paper presented a new inference rule, SAFE, for proving arbitrary general safety properties expressed in linear-time temporal logic. It generalizes the usual inference rule for proving invariance properties. The intermediate assertions needed for the application of the rule can be found in two dual ways: as least and greatest fixedpoints respectively. As this is usually not possible, we applied and developed tools from abstract interpretation to find approximations. In the case studies we developed new methodologies for abstract interpretation of parameterized programs.

The approach taken for the generation of invariants and intermediary assertions was to reduce a verification problem to domains that admit specialized and efficient constraint-solving methods. In prescribing general methodologies for such reductions we have not attempted to use special control features of the analyzed systems. Instead we outline how new automatic invariant generation and strengthening methods can be obtained by finding a suitable approximation domain, a suitable constraint language over this domain, and sufficiently powerful constraint-solving procedures for this constraint language. A main challenge for further work is to identify useful domains that admit efficient constraint-solvers.

Acknowledgement

We thank Martín Abadi, Luca de Alfaro, Henny Sipma, and Tomás Uribe for their many helpful comments and suggestions. We would also like to thank the two anonymous referees for their helpful suggestions.

References

[BBC+95] N. Bjørner, A. Browne, E. Chang, M. Colón, A. Kapur, Z. Manna, H.B. Sipma, and T.E. Uribe. STeP: The Stanford Temporal Prover, User's Manual. Technical Report STAN-CS-TR-95-1562, Computer Science Department, Stanford University, Nov. 1995.

[BBM95] N. Bjørner, A. Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. In 1st *Intl. Conf. on Principles and Practice of Constraint Programming*, vol. 976 of *LNCS*, pages 589–623. Springer-Verlag, Sept. 1995.

[BLS96] S. Bensalem, Y. Lakhnech, and H. Saidi. Powerful Techniques for the

Automatic Generation of Invariants. In *Proc. 8^{th} Intl. Conference on Computer Aided Verification*, LNCS. Springer-Verlag, 1996. To appear.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In 4^{th} *ACM Symp. Princ. of Prog. Lang.*, pages 238–252. ACM Press, 1977.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among the variables of a program. In 5^{th} *ACM Symp. Princ. of Prog. Lang.*, pages 84–97, Jan. 1978.

[CP93] R. Chadha and D. A. Plaisted. On the mechanical derivation of loop invariants. *Journal of Symbolic Computation*, 15(5):705–744, 1993.

[DW95] D. Dill and H. Wong-Toi. Verification of real-time systems by successive over and under approximation. In P. Wolper, editor, *Proc. 7^{th} Intl. Conference on Computer Aided Verification*, LNCS, pages 409–422. Springer-Verlag, 1995.

[GW75] S. M. German and B. Wegbreit. A Synthesizer of Inductive Assertions. *IEEE transactions on Software Engineering*, 1(1):68–75, March 1975.

[Har84] D. Harel. Statecharts: A visual approach to complex systems. Technical Report CS84-05, Dept. of Applied Mathematics, Weizmann Institute of Science, 1984.

[Hei92] N. Heintze. *Set Based Program Analysis*. PhD thesis, Carnegie Mellon University, 1992.

[HH95] T.A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. In P. Wolper, editor, *Proc. 7^{th} Intl. Conference on Computer Aided Verification*, vol. 939 of *LNCS*, pages 225–238. Springer-Verlag, 1995.

[HRP94] N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximations. In 1^{st} *Intl. Static Analysis Symp.*, vol. 864 of *LNCS*, pages 223–237. Springer-Verlag, Sept. 1994.

[JM94] J. Jaffar and M.J. Maher. Constraint Logic Programming: a survey. *Journal of Logic Programming*, 19-20(3):503–81, 1994.

[Kar76] M. Karr. Affine relationships among variables of a program. *Acta Informatica*, 6:133–151, 1976.

[KM76] S. Katz and Z. Manna. Logical analysis of programs. *Communications of the ACM*, 19(4):188–206, April 1976.

[Lau73] K. Lautenbach. Exacte Bedingungen der Lebendigkeit für eine Klasse von Petri-Netzen (*Precise Conditions for Liveness of a Class of Petri-Nets*). *St. Augustin, GMD Bonn*, 82, 1973.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.