# Better Under-approximation of Programs by Hiding Variables

Thomas Ball[1] and Orna Kupferman[2]

[1] Microsoft Research, One Microsoft way, Redmond, WA, 98052, USA.
Email: tball@microsoft.com, URL: research.microsoft.com/∼tball
[2] Hebrew University, School of Eng. and Comp. Sci., Jerusalem 91904, Israel.
Email: orna@cs.huji.ac.il, URL: www.cs.huji.ac.il/∼orna

**Abstract.** Abstraction frameworks use under-approximating transitions in order to prove existential properties of concrete systems. Under-approximating transitions refer to the concrete states that correspond to a particular abstract state in a universal manner. For example, there is a *must* transition from abstract state $a$ to abstract state $a'$ only if all the concrete states in $a$ have successors in $a'$.
The universal nature of under-approximating transitions makes them closed under transitivity. Consequently, reachability queries about the concrete system, which have applications in falsification and testing, can be answered by reasoning about its abstraction. On the negative side, the universal nature of under-approximating transitions makes them dependent on all the variables of the program. The abstraction, on the other hand, often hides some of the variables. Since the universal quantification in must transitions ranges over all variables, this often prevents the abstraction from associating a must transition with statements that refer to hidden variables.
We introduce and study *partitioned-must* transitions. The idea is to partition the program variables to relevant and irrelevant ones, and restrict the universal quantification inside must transitions to the relevant variables. Usual must transitions are a special case of partitioned-must transitions in which all variables are relevant. Partitioned-must transitions exist in many realistic settings in which usual must transitions do not exist. As we show, they retain the advantages of must transitions: they are closed under transitivity, their calculation can be automated, and the three-valued semantics induced by usual must transitions is refined to a multi-valued semantics that takes into an account the set of relevant variables.

## 1 Introduction

Abstraction frameworks [CC77] generally use *over-approximation* to check safety properties. If a safety property holds in the abstract (over-approximate) system then it holds in the concrete system that it abstracts. However, if the safety property does not hold in the abstract system, we do not know if the concrete system violates the safety property.

Since the ideal goal of proving a system correct involves many obstacles, the primary use of formal methods nowadays is *falsification*. There, as in *testing*, the goal is to detect errors, rather than to prove correctness. In the falsification setting, we are interested in using abstractions based on *under-approximation*. This allows us to prove that

if a safety property does not hold in the abstract system then it does not hold in the concrete system. Our investigations are based on modal transition systems (MTS) [LT88], which combine both overapproximation and under-approximation. Traditional MTSs have two types of transitions: *may* (over-approximating transitions) and *must* (under-approximating transitions).

A must transition from an abstract state $a$ to an abstract state $a'$ implies that for all concrete states $c$ that correspond to $a$ there is a successor concrete state $c'$ that corresponds to $a'$. The importance of must transitions comes from the fact they are closed under transitivity: if there is a sequence of must transitions from $a$ to $a'$, we can conclude that all concrete states $c$ that correspond to $a$ can reach some concrete state $c'$ that corresponds to $a'$.

Unfortunately, must transitions are very fragile with respect to updates of *irrelevant variables*. To see this, consider, for example, two abstract states $(x > 6)$ and $(x > 8)$. Assume that the statement `if y=0 then {x:=x+4;read(y)}` is executed at $(x > 6)$. Since the abstraction ignores the variable $y$, and not all the concrete states in $(x > 6)$ have $y = 0$, there is no must transition from $(x > 6)$ to $(x > 8)$. For example, the concrete state $\langle 7, 1\rangle$ has no successor state in $(x > 8)$. Current abstraction frameworks would therefore include a may transition from $(x > 6)$ to $(x > 8)$, and are likely to end up refining these states with predicates that refer to $y$.

This is needlessly too weak. A may transition only guarantees reachability for existentially quantified values of $x$ and $y$: there exist values of $x$ and $y$ satisfying $(x > 6)$ for which there exist successor values satisfying $(x > 8)$. The actual situation, however, has a richer type of reachability, in which we can quantify the value of $x$ universally and quantify only the value of $y$ existentially. In this work we introduce and study *partitioned must* transitions, which enable us to capture situations as above.

In order to understand our partitioned-must transitions, let us first recall earlier efforts to extend the usefulness of must transitions. Consider again the abstract states $(x > 8)$ and $(x > 6)$, and assume that the statement `x:=x-4` is executed at $(x > 8)$. Since there are concrete states satisfying $x > 8$ (namely $x = 9$ and $x = 10$) for which the assignment statement results in a successor state that does not satisfy $x > 6$, the abstract transition from $(x > 8)$ to $(x > 6)$ is not a must transition. Augmenting MTSs with hyper-must transitions [LX90,SG04] does not help in this setting either (and is orthogonal to the contribution we describe here).

Such cases motivated the introduction of $must^-$ transitions [Bal04]. A $must^-$ transition from $a$ to $a'$ implies that for all concrete states $c'$ that correspond to $a'$ there is a concrete predecessor state $c$ that corresponds to $a$. In the above example, there is a $must^-$ transition from $(x > 8)$ to $(x > 6)$. Like must transitions (let us refer to them in the sequel as $must^+$ transitions), $must^-$ transitions are closed under transitivity, and as argued in [Bal04,BKY05], they are often useful in cases $must^+$ transitions do not exist.

While $must^-$ transitions are helpful in scenarios as above, they do not address the fragility of must transitions with respect to updates of irrelevant variables. In particular, in our earlier example, of $(x > 6)$ `if y=0 then {x:=x+4;read(y)}` $(x > 8)$, there is no $must^-$ transition from $(x > 6)$ to $(x > 8)$, as there are concrete states satisfying $x > 8$ (namely $\langle 9, 0\rangle$ and $\langle 10, 0\rangle$) that are not reachable from any concrete state satisfying $x > 6$. Moreover, while $must^-$ transitions came to the rescue in the $(x > 8)$

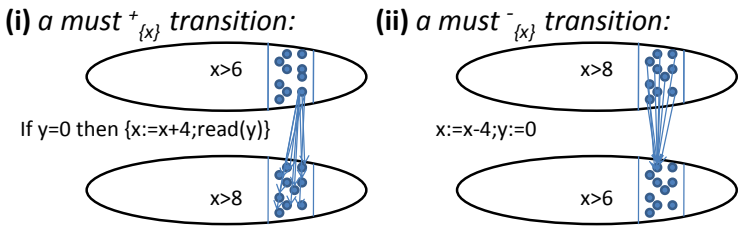


**Fig. 1.** Partitioned-must transitions.

`x:=x-4` $(x > 6)$ example, they are no longer useful when we add irrelevant variables. Assume, for example, that the statement executed in $(x > 8)$ is `x:=x-4;y:=0`. Since the abstraction ignores the variable $y$ and not all the concrete states in $(x > 6)$ have $y = 0$, there are concrete states in $(x > 6)$, say $\langle 10, 1\rangle$, that do not have a predecessor in $(x > 8)$. Accordingly, there is no $must^-$ transition in the new setting.

As hinted earlier, the idea behind our partitioned-must transitions is to restrict the universal nature of must transitions to a subset of the variables. Given a set $X$ of *relevant variables*, we can partition the state space of the concrete system to equivalence classes such that states in the same class agree on the values of the variables in $X$. Consider again the $(x > 6)$ `if y=0 then {x:=x+4;read(y)}` $(x > 8)$ example (see Figure 1 (i)). We argue that if we restrict attention to the set $X = \{x\}$ of relevant variables, then there is a partitioned $must^+$ transition from $(x > 6)$ to $(x > 8)$ in the following sense. For every concrete state $\langle x, y\rangle$ in $(x > 6)$, there is a concrete state $\langle x', y'\rangle$ in $(x > 8)$ such that all the states in the equivalence class of $\langle x', y'\rangle$ have a predecessor in the equivalence class of $\langle x, y\rangle$. Indeed, $x' = x + 4$ is such that all the states in $\{x + 4\} \times \mathbb{N}$ are reachable from the state $\langle x, 0\rangle$, which is in the equivalence class of $\langle x, y\rangle$.

In general, we say that there is a $pmust^+_X$ transition from $a$ to $a'$ only if for every concrete state $c$ that corresponds to $a$ there is a concrete state $c'$ that corresponds to $a'$ such that there is a $must^-$ transition from the restriction of $a$ to the equivalence class of $c$ to the restriction of $a'$ to the equivalence class of $c'$. Dually, there is a $pmust^-_X$ transition from $a$ to $a'$ only if for every concrete state $c'$ that corresponds to $a'$ there is a concrete state $c$ that corresponds to $a$ such that there is a $must^+$ transition from the restriction of $a$ to the equivalence class of $c$ to the restriction of $a'$ to the equivalence class of $c'$. For example (see Figure 1 (ii)), in the $(x > 8)$ `x:=x-4;y:=0` $(x > 6)$ setting, there is a $pmust^-_{\{x\}}$ transition from $(x > 8)$ to $(x > 6)$.

In the paper, we define partitioned-must transitions, characterize settings in which they are useful, and study their theoretical properties. As we show, while partitioned-must transitions exist in many realistic settings in which usual must transitions do not exist, they retain the advantages of must transitions: they are closed under transitivity, their calculation can be automated, and the three-valued semantics induced by usual must transitions is refined to a multi-valued semantics that takes into an account the set of relevant variables.

## 2 Preliminaries

*Programs and Concrete Transition Systems* Consider a program $P$. Let $V$ be the set of variables appearing in the program and variables that encode the program counter ($pc$), and let $D$ be the domain of all variables (for technical simplicity, we assume that all variables are over the same domain). We model $P$ by a concrete transition system in which each state is labeled by a valuation in $D^{|V|}$.

A *concrete transition system* (CTS) is a tuple $C = \langle S_C, I_C, \longrightarrow_C \rangle$, where $S_C$ is a (possibly infinite) set of states, $I_C \subseteq S_C$ is a set of initial states, $\longrightarrow_C \subseteq S_C \times S_C$ is a total transition relation. Let $c \longrightarrow_C{}^* c'$ denote that state $c'$ is reachable from state $c$ via a path of transitions.

*Predicate Abstraction* Let $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$ be a set of predicates (quantifier-free formulas of first-order logic) over the program variables $V$. In the CTS of the program, each concrete state $c$ corresponds to a valuation of $V$. Given a program state $c$ and formula $\phi$, let $c \models \phi$ indicate that formula $\phi$ is true in state $c$ ($c$ is a model of $\phi$). For a set $a \subseteq \Phi$ and an assignment $c \in D^V$, we say that $c$ *satisfies* $a$ iff $c \models \bigwedge_{\phi_i \in a} \phi_i$.

In predicate abstraction, we merge a set of concrete states into a single abstract state, which is defined by means of a subset of the predicates. Thus, an abstract state is given by a set of predicates $a \subseteq \Phi$.[1] We sometimes represent $a$ by a formula, namely the conjunction of predicates in $a$. For example, if $a = \{(x \geq y), (0 \leq x < n)\}$ then we also represent $a$ by the formula $(x \geq y) \wedge (0 \leq x < n)$. We define the set of concrete states corresponding to $a$, denoted $\gamma(a)$, as all the states $c$ that satisfy $a$; that is, $\gamma(a) = \{c \mid c \models a\}$.

*May and Must Transitions* Given a CTS and its (predicate) abstraction via a set of predicates $\Phi$, a *modal transition system* (MTS) contains three kinds of abstract transitions between abstracts states $a$ and $a'$ ($a, a' \subseteq \Phi$, and we assume that $\Phi$ is clear from the context):

- $may(a, a')$ if there is $c \in \gamma(a)$ and a $c' \in \gamma(a')$, such that $c \longrightarrow_C c'$.
- $must^+(a, a')$ only if for every $c \in \gamma(a)$, there is $c' \in \gamma(a')$ such that $c \longrightarrow_C c'$.
- $must^-(a, a')$ only if for every $c' \in \gamma(a')$, there is $c \in \gamma(a)$ such that $c \longrightarrow_C c'$.

While $may$ transitions over-approximate the transitions of the CTS, both types of $must$ transitions under-approximate it. It is not hard to see that must transitions are closed under transitivity, and can therefore be used to prove reachability in the concrete system. Formally, if there is a sequence of $must^+$ transitions from $a$ to $a'$, denoted $must^{+*}(a, a')$, then for all $c \in \gamma(a)$, there is $c' \in \gamma(a')$ such that $c \longrightarrow_C{}^* c'$. The same holds for $must^-$. Formally, if there is a sequence of $must^-$ transitions from $a$ to $a'$, denoted $must^{-*}(a, a')$, then for all $c' \in \gamma(a')$, there is $c \in \gamma(a)$ such that $c \longrightarrow_C{}^* c'$.

On the other hand, may transitions are not transitive. Indeed, it may be the case that $may(a, a'), may(a', a'')$ and still for all $c \in a$ and $c'' \in a''$, we have $c \not\longrightarrow_C{}^* c'$.

[1] In the full generality of predicate abstraction, an abstract state is represented by a set of set of predicates (that is a, disjunction of conjunction of predicates). All our results hold for the more general setting.

*Weakest Preconditions and Strongest Postconditions* In many applications of predicate abstraction, $\Psi$ includes a predicate for the program counter. Accordingly, each abstract state is associated with a location of the program, and thus it is also associated with a statement. For a statement $s$ and a predicate $e$ over $V$, the *weakest precondition* $\mathrm{WP}(s, e)$ and the *strongest postcondition* $\mathrm{SP}(s, e)$ are defined as follows [Dij76]:

- The execution of $s$ from every state that satisfies $\mathrm{WP}(s, e)$ results in a state that satisfies $e$, and $\mathrm{WP}(s, e)$ is the weakest predicate for which the above holds.
- The execution of $s$ from a state that satisfies $e$ results in a state that satisfies $\mathrm{SP}(s, e)$, and $\mathrm{SP}(s, e)$ is the strongest predicate for which the above holds.

Must transitions can be computed automatically using weakest preconditions and strongest postconditions. Indeed, statement $s$ induces the transition $must^+(a, a')$ iff $a \Rightarrow \mathrm{WP}(s, a')$, and induces the transition $must^-(a, a')$ iff $a' \Rightarrow \mathrm{SP}(s, a)$.

## 3 Partitioned-Must Transitions

Recall that we consider programs with variables $V$ over the domain $D$. For a set $X \subseteq V$, we define a relation $\sim_X \subseteq D^V \times D^V$ between concrete states such that $c \sim_X c'$ iff $c$ and $c'$ agree on the values of the variables in $X$. For a concrete state $c$, let $[c]_X = \{c' : c \sim_X c'\}$; that is, $[c]_X$ is the set of concrete states that agree with $c$ on the values of the variables in $X$.

We are now ready to introduce partitioned-must transitions. The idea is to partition the variables of the program to relevant ($X$) and irrelevant ($V \setminus X$) variables and restrict the universal quantification in must transitions to range over the equivalence classes of $\sim_X$. Formally, we have the following.

**Definition 1.** *Consider two abstract states $a$ and $a'$, and a set $X \subseteq V$.*

1. *There is a $pmust^+_X$ transition from $a$ to $a'$, denoted $pmust^+_X(a, a')$, only if for all $c \in \gamma(a)$ there is $c' \in \gamma(a')$ such that $must^-([c]_X \wedge a, [c']_X \wedge a')$.*
2. *There is a $pmust^-_X$ transition from $a$ to $a'$, denoted $pmust^-_X(a, a')$, only if for all $c' \in \gamma(a')$ there is $c \in \gamma(a)$ such that $must^+([c]_X \wedge a, [c']_X \wedge a')$.*

**Example 1** Let us go back to the examples discussed in Section 1 and review them formally. Consider the transition $(x > 6)$ `if y=0 then {x:=x+4;read(y)}` $(x > 8)$. Assume that $V = \{x, y\}$, and let the domain of both variables be $\mathbb{N}$. There is a $pmust^+_{\{x\}}$ transition from $(x > 6)$ to $(x > 8)$. Indeed, for all concrete states $\langle x, y\rangle \in \gamma(x > 6)$, there exists the concrete state $\langle x + 4, y\rangle \in \gamma(x > 8)$ for which $must^-([\langle x, y\rangle]_{\{x\}}, [\langle x + 4, y\rangle]_{\{x\}})$. To see the latter, note that $[\langle x + 4, y\rangle]_{\{x\}} = \{x + 4\} \times \mathbb{N}$, and each state in $\{x + 4\} \times \mathbb{N}$ is reachable from $\langle x, 0\rangle$, which is in $[\langle x, y\rangle]_{\{x\}}$. Thus, the partition to $\{x\}$ and $\{y\}$ circumvents the need to refer to the value of $y$ in the destination state.

Consider the transition $(x > 8)$ `x:=x-4; y:=0` $(x > 6)$. There is a $pmust^-_{\{x\}}$ transition from $(x > 8)$ to $(x > 6)$. Indeed, for all concrete states $\langle x, y\rangle \in \gamma(x > 6)$, there exists the concrete state $\langle x + 4, y\rangle \in \gamma(x > 8)$ for which $must^+([\langle x + 4, y\rangle]_{\{x\}}, [\langle x, y\rangle]_{\{x\}})$. To see the latter, note that $[\langle x + 4, y\rangle]_{\{x\}} = \{x + 4\} \times \mathbb{N}$. Executing the statement `x:=x-4;y:=0` from a state in $\{x + 4\} \times \mathbb{N}$ results in the state

$\langle x, 0\rangle$, which is in $[\langle x, y\rangle]_{\{x\}} = \{x\} \times \mathbb{N}$. Thus, also here, the partition to $\{x\}$ and $\{y\}$ circumvents the need to refer to the value of $y$ in the destination state.

□

Example 1 demonstrates cases in which $must^+$ and $must^-$ transitions do not exist but partitioned-must transitions do exist. Below we characterize such cases in general:

- The abstraction refers to variables in $X$ only, and the statement involves an assignment to variables in $V \setminus X$. Here, there is no $must^-$ transition, but there is a $pmust^-_X$ transition. The example $(x > 8)$ `x:=x-4;y:=0` $(x > 6)$ is emblematic of this case.
- The abstraction refers to variables in $X$ only, and the statement involves guards on the variables in $V \setminus X$. The range of the guarded variables in the post state is not restricted to these that satisfy the guard (due to nondeterminism or the infiniteness of the domain). Here, there is no $must^+$ transitions, but there is a $pmust^+_X$ transition. The example $(x > 6)$ `if y=0 then {x:=x+4; read(y)}` $(x > 8)$ is emblematic of this case. An example of a similar nature but with more restricted nondeterminism is

  $$(x > 6)\ \texttt{if y is odd then \{x:=x+4;(skip|y:=y-1)\}}\ (x > 8).$$

  Here, not all concrete states $\langle x, y\rangle \in \gamma(x > 6)$ have a successor in $\gamma(x > 8)$, but for all concrete states $\langle x, y\rangle \in \gamma(x > 6)$, there exists the concrete state $\langle x + 4, y\rangle \in \gamma(x > 8)$ for which $must^-([\langle x, y\rangle]_{\{x\}}, [\langle x + 4, y\rangle]_{\{x\}})$. Indeed, the nondeterministic assignment guarantees that all values of $y$ have a pre-state with an odd value. As a last example for this case, consider

  $$(x > 6)\ \texttt{if y>=10 then \{x:=x+4; y:=y-10\}}\ (x > 8).$$

  Here, the program is deterministic, and still, the fact $\mathbb{N}$ is infinite, thus $y$ can take any value that is greater than or equal to 0, implies that all values of $y$ in the post-state are covered by values greater than or equal to 10 in the pre-state.
- The abstraction refers to all variables, but for these in $X$, it over-approximates the value in the post-state and for these in $V \setminus X$ it over-approximates the value in the pre-state. While there are no $must^+$ or $must^-$ transitions, there are $pmust^+_X$ and $pmust^-_{V \setminus X}$ transitions. A typical example for this case is

  $$(x > 6, y > 8)\ \texttt{x:=x+4;y:=y-4}\ (x > 8, y > 6).$$

We now show that partitioned-must transitions are closed under transitivity. For two abstract states $a$ and $a'$ and a set of variables $X \subseteq V$, we use $pmust^{+\,*}_X(a, a')$ to indicate that there is a (possibly empty) sequence of $pmust^+_X$ transitions from $a$ to $a'$. Formally, there are $a_1, a_2, \ldots, a_n$ such that $a = a_1$, $a_n = a'$, and for all $1 \leq i < n$, we have that $pmust^+_X(a_i, a_{i+1})$. The notation $pmust^{-\,*}_X(a, a')$ is defined similarly as the transitive closure of $pmust^-_X$ transitions.

**Proposition 1. [transitive closure]** *Consider two abstract states $a$ and $a'$, and a set $X \subseteq V$.*

1. *If $pmust^{+}_{X}{}^{*}(a, a')$, then for all $c \in \gamma(a)$ there exists $c' \in \gamma(a')$ such that $must^{-}{}^{*}([c]_X \wedge a, [c']_X \wedge a')$.*
2. *If $pmust^{-}_{X}{}^{*}(a, a')$, then for all $c' \in \gamma(a')$ there exists $c \in \gamma(a)$ such that $must^{+}{}^{*}([c]_X \wedge a, [c']_X \wedge a')$.*

**Proof:** We prove the forward case, the backwards case is dual. Assume that $pmust^{+}_{X}{}^{*}(a, a')$. Let $a_1, a_2, \ldots, a_n$ be such that $a = a_1$, $a' = a_n$, and for all $1 \leq i < n$, we have $pmust^{+}_{X}(a_i, a_{i+1})$. We prove that for all $c_1 \in \gamma(a_1)$ there is $c_n \in \gamma(a_n)$ such that $must^{-}{}^{*}([c_1]_X \wedge a_1, [c_n]_X \wedge a_n)$. The proof proceeds by induction on the length of the sequence of transitions (i.e., $n - 1$). If $n = 1$, the sequence is empty and the requirement follows from the definition of $*$. Assume that the claim holds for sequences of length $n$, and consider a sequence of length $n + 1$. By the induction hypothesis, for all $c_1 \in \gamma(a_1)$ there is $c_n \in \gamma(a_n)$ such that $must^{-}{}^{*}([c_1]_X \wedge a_1, [c_n]_X \wedge a_n)$. Since $pmust^{+}_{X}{}^{*}(a_n, a_{n+1})$, then for all $c_n \in \gamma(a_n)$ there is $c_{n+1} \in \gamma(a_{n+1})$ such that $must^{-}([c_n]_X \wedge a_n, [c_{n+1}]_X \wedge a_{n+1})$. By the transitivity of $must^{-}$, we can conclude that $must^{-}{}^{*}([c_1]_X \wedge a_1, [c_{n+1}]_X \wedge a_{n+1})$.

□

Traditional $must^{+}$ and $must^{-}$ transitions can be viewed as the two *polar* cases of partitioned must transitions. Formally, we have the following:

**Proposition 2.** *For all abstract states $a$ and $a'$, the following hold.*

1. *$must^{+}(a, a')$ iff $pmust^{+}_{V}(a, a')$ iff $pmust^{-}_{\emptyset}(a, a')$.*
2. *$must^{-}(a, a')$ iff $pmust^{+}_{\emptyset}(a, a')$ iff $pmust^{-}_{V}(a, a')$.*

**Proof:** We prove the forward case, the backwards case is dual. When $X = V$, the relation $\sim_X$ relates each state only with itself. Thus, $pmust^{+}_{V}(a, a')$ iff for all $c \in \gamma(a)$ there is $c' \in \gamma(a')$ such that $must^{-}(c, c')$. Since for concrete states, we have that $must^{-}(c, c')$ coincides with $c \longrightarrow_C c'$, it follows that $must^{+}(a, a')$ iff $pmust^{+}_{V}(a, a')$.

When $X = \emptyset$, the relation $\sim_X$ relates all states in $D^V$. Thus, the condition $must^{+}([c]_X \wedge a, [c']_X \wedge a')$ is independent of $c$ and $c'$ and is equivalent to $must^{+}(a, a')$. □

*Remark 1.* There are abstract states $a$ and $a'$ such that $must^{+}(a, a')$ and $must^{-}(a, a')$ and the only sets $X$ for which $pmust^{+}_{X}(a, a')$ or $pmust^{-}_{X}(a, a')$ are the polar ones.

To see this, consider Boolean variables $x$ and $y$ and let $a = a' = true$. The transition `if x=y then skip else swap(x,y)` induces a $must^{+}$ as well as a $must^{-}$ transition from $a$ to $a'$. The four possible partitions of $\{x, y\}$ and the partitioned transitions they induce are described in Figure 2. As described there, there is no $pmust^{+}_{\{x\}}$, $pmust^{-}_{\{x\}}$, $pmust^{+}_{\{y\}}$, or $pmust^{-}_{\{y\}}$ transition from $a$ to $a'$.

**Example 2** We demonstrate the usefulness of partitioned-must transitions with a variant of the well-known algorithm for calculating the greatest-common-divisor of two positive integers. Consider the function `gcd` described in Figure 3.

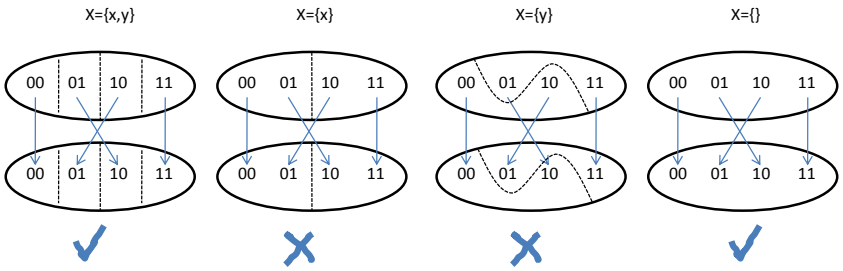


**Fig. 2.** Existence and nonexistence of a partitioned-must transition.

```
gcd(x,y) {
 (1)    assume(x>0);
 (2)    assume(y>0);
 (3)    int t:=0;
 (4)    while (x!=y) do if (x>y) then x:=x-y;t:=t+1
                                 else y:=y-x
}
```

**Fig. 3.** The function gcd.

In addition to the variables $x$ and $y$ whose gcd is calculated, the function maintains a variable $t$ that counts the number of iterations in which $x > y$. Consider an abstraction that refers to $x$, $y$, and the program counter $pc$. Consider the abstract state $a = (pc = 4 \wedge x > 0 \wedge y > 0 \wedge x \neq y)$. We would like to show that all values of $x, y$, and $pc$ that satisfy $a$ are successors of other values that satisfy $a$. Thus, whenever we are in the loop with $x \neq y$, we have a predecessor in the loop with $x \neq y$. An attempt to prove the above with $must^-$ transitions fails: since the abstraction ignores the value of $t$, concrete states that satisfy $a$ and in which $t = 0$ may not have a predecessor that satisfy $a$ (as they may be reachable only from states visited before the execution of the loop, and in which $pc \neq 4$). Hiding the variable $t$, however, we can prove the above by showing that $pmust^-_{\{x,y,pc\}}(a, a)$. To see the latter, observe that for all $\langle x, y, t, pc\rangle \in \gamma(a)$, we have that $must^+([\langle x + y, y, t, pc\rangle]_{\{x,y\}}, [\langle x, y, t, pc\rangle]_{\{x,y\}})$. Indeed, satisfying $a$ guarantees that $pc = 4$, and the execution of the statement in $pc = 4$ from the values $\langle x+y, y, t, 4\rangle$, which satisfy $a$, results in values $\langle x, y, t, 4\rangle$.

Now, consider the abstract state $b = (pc = 4 \wedge x > 0 \wedge y > 0 \wedge x = y)$, for which all corresponding concrete states cause the while loop to terminate. We would like to show that all values of $x$, $y$, and $pc$ that satisfy $b$ are successors of other values that satisfy $a$. Thus, whenever we are in the loop with $x = y$, we have a predecessor in the loop with $x \neq y$. Again, an attempt to prove the above with $must^-$ transitions fails: since the abstraction ignores the value of $t$, concrete states that satisfy $b$ and in which $t = 0$ may not have a predecessor that satisfies the $pc = 4$ conjunct in $a$. Hiding the variable $t$, however, we can prove the above by showing that $pmust^-_{\{x,y,pc\}}(a, b)$. To see the latter, observe that for all $\langle x, y, t, pc\rangle \in \gamma(b)$, we have that $must^+([\langle x, 2x, t, pc\rangle]_{\{x,y\}}, [\langle x, y, t, pc\rangle]_{\{x,y\}})$. Finally, by the transitivity of

$pmust^-_X$ transitions, we can conclude that whenever we are about to leave the loop with $x = y$, and for any desired iteration count $i$, we can go back $i$ transitions and stay in the loop with $x \neq y$. □

## 4 Calculation of Partitioned-Must Transitions

As Example 2 shows, the calculation of partitioned-must transitions may not be easy. In this section we show that this calculation can be automated. We start with $pmust^+_X$ transitions.

**Theorem 3.** *Consider two abstract states $a$ and $a'$ and a set $X \subseteq V$. Let $s$ be the statement executed in $a$. The following are equivalent.*

1. $pmust^+_X(a, a')$.
2. *For all $c \in \gamma(a)$ there is $c' \in \gamma(a')$ such that $([c']_X \wedge a') \Rightarrow \mathrm{SP}(s, [c]_X \wedge a)$.*
3. *For all $c \in \gamma(a)$, there is an equivalence class $\theta$ of $\sim_X$ such that $\theta \wedge a'$ is satisfiable and $(\theta \wedge a') \Rightarrow \mathrm{SP}(s, [c]_X \wedge a)$.*

**Proof:** We prove that both (1) and (3) are equivalent to (2). We start by proving that $(1) \leftrightarrow (2)$. By Definition 1, $pmust^+_X(a, a')$ iff for all $c \in \gamma(a)$ there is $c' \in \gamma(a')$ such that $must^-([c]_X \wedge a, [c']_X \wedge a')$. By the definition of $must^-$ transitions, the latter holds iff $([c']_X \wedge a') \Rightarrow \mathrm{SP}(s, [c]_X \wedge a)$, and we are done.

It is left to prove that $(2) \leftrightarrow (3)$. Assume first that (2) holds. Thus, for all $c \in \gamma(a)$ there is $c' \in \gamma(a')$ such that $([c']_X \wedge a') \Rightarrow \mathrm{SP}(s, [c]_X \wedge a)$. Then, given $c \in \gamma(a)$, the set $\theta = [c']_X$ is an equivalence class of $\sim_X$ such that $\theta \wedge a'$ is satisfiable (say, by $c'$), and $(\theta \wedge a') \Rightarrow \mathrm{SP}(s, [c]_X \wedge a)$. Assume now that (3) holds. Thus, for all $c \in \gamma(a)$ there is an equivalence class $\theta$ of $\sim_X$ such that $\theta \wedge a'$ is satisfiable and $(\theta \wedge a') \Rightarrow \mathrm{SP}(s, [c]_X \wedge a)$. Let $c'$ be such that $c'$ satisfies $\theta \wedge a'$. Since $\theta$ is an equivalence class of $\sim_X$, we have that $[c']_X \Rightarrow \theta$. Hence, $c'$ is such that $c' \in \gamma(a')$ and $([c']_X \wedge a') \Rightarrow \mathrm{SP}(s, [c]_X \wedge a)$. □

We can now use Theorem 3 to describe a first-order logic formula that is valid iff the conditions for the existence of a $pmust^-_X$ transition are satisfied. Describing the formula, we use $\boldsymbol{x}$ and $\boldsymbol{y}$ (possibly primed) to denote the variables in $X$ and $V \setminus X$, respectively. For a predicate $a$ over $V$, we use $a(\boldsymbol{x}, \boldsymbol{y})$ to indicate that the assignment of the variables in $V$ (described in $\boldsymbol{x}$ and $\boldsymbol{y}$ together) satisfy $a$. Finally, when we use $\boldsymbol{x}$ as a predicate, it is satisfied by assignments to $V$ that agree with $\boldsymbol{x}$ on the variables in $X$.

**Proposition 3.** *There is a $pmust^+_X$ transition from $a$ to $a'$ only if the following formula is valid.*

$$\forall \boldsymbol{x} \forall \boldsymbol{y} [a(\boldsymbol{x}, \boldsymbol{y}) \rightarrow \exists \boldsymbol{x'} ((\exists \boldsymbol{y'}.a'(\boldsymbol{x'}, \boldsymbol{y'})) \wedge (\forall \boldsymbol{y'}.(a'(\boldsymbol{x'}, \boldsymbol{y'}) \rightarrow \mathrm{SP}(s, \boldsymbol{x} \wedge a)))].$$

**Proof:** The formula states that for all states $c \in \gamma(a)$ (these are the universally quantified variables in $\boldsymbol{x}$ and $\boldsymbol{y}$, when they satisfy the left-hand side of the $a(\boldsymbol{x}, \boldsymbol{y}) \rightarrow \ldots$ implication), there is an equivalence class of $\sim_X$ (these are the existentially quantified variables in $\boldsymbol{x'}$) that satisfies the condition in item (3) of Theorem 3: the intersection of the equivalence class with $a'$ is not empty (there is an assignment $\boldsymbol{y'}$ to the variables in $V \setminus X$ such that $a'(\boldsymbol{x'}, \boldsymbol{y'})$), and every assignment in the intersection (that is, every $\boldsymbol{y'}$ that is combined with $\boldsymbol{x'}$ and for which $a'(\boldsymbol{x'}, \boldsymbol{y'})$) satisfies $\mathrm{SP}(s, \boldsymbol{x} \wedge a)$. □

We now describe a similar reasoning for $pmust^-_X$ transitions. The proof is similar to the one detailed in the proofs of Theorem 3 and Proposition 3.

**Theorem 4.** *Consider two abstract states $a$ and $a'$ and a set $X \subseteq V$. Let $s$ be the statement executed in $a$. The following are equivalent.*

1. $pmust^-_X(a, a')$.
2. *For all $c' \in \gamma(a')$ there is $c \in \gamma(a)$ such that $([c]_X \wedge a) \Rightarrow \mathrm{WP}(s, [c']_X \wedge a')$.*
3. *For all $c' \in \gamma(a')$ there is an equivalence class $\theta$ of $\sim_X$ such that $\theta \wedge a$ is satisfiable and $(\theta \wedge a) \Rightarrow \mathrm{WP}(s, [c']_X \wedge a')$.*

**Proposition 4.** *There is a $pmust^-_X$ transition from $a$ to $a'$ only if the following formula is valid.*

$$\forall \boldsymbol{x'} \forall \boldsymbol{y'} [a'(\boldsymbol{x'}, \boldsymbol{y'}) \rightarrow \exists \boldsymbol{x}((\exists \boldsymbol{y}.a(\boldsymbol{x}, \boldsymbol{y})) \wedge (\forall \boldsymbol{y}.(a(\boldsymbol{x}, \boldsymbol{y}) \rightarrow \mathrm{WP}(s, \boldsymbol{x'} \wedge a'))))].$$

When the predicates of the abstraction contain only variables appearing in $X$, reasoning is simplified. We discuss this case in Section 6, where we also show the simplified version of the formulas described in Propositions 3 and 4 for the general case. In particular, in Example 6 there, we describe the automation of the reasoning required for the gcd function discussed in Example 2.

## 5 Applications

In this section we discuss applications of partitioned-must transitions. Essentially, our applications are these in which one is interested in *weak reachability* in the abstract system. For two abstract states $a$ and $a'$, we say that $a'$ is weakly reachable from $a$ iff there are concrete state $c \in \gamma(a)$ and $c' \in \gamma(a')$ such that $c'$ is reachable from $c$. While weak reachability quantifies the states in $\gamma(a)$ and $\gamma(a')$ existentially, we cannot use may transitions in order to detect it, as may transitions are not closed under transitivity. Thus, the way to go is to check whether $must^{+^*}(a, a')$ or $must^{-^*}(a, a')$. The fragility of must transitions with respect to irrelevant variables can then prevent the detection of weak reachability, and we suggest to use partitioned-must transition instead. Below we detail the applications in falsification and verification of temporal-logic specifications.

### 5.1 Linear-time falsification

In *linear-time model checking*, we check whether all the computations of a given program $P$ satisfy a specification $\psi$, say an LTL formula. In the automata-theoretic approach to model checking [Kur94,VW94], one constructs an automaton $\mathcal{A}_{\neg\psi}$ for the negation of $\psi$. The automaton $\mathcal{A}_{\neg\psi}$ is usually a nondeterministic Büchi automaton, where a run is accepting iff it visits a set of designated states infinitely often. The program $P$ is faulty with respect to $\psi$ if the product of $\mathcal{A}_{\neg\psi}$ with the program contains a fair path – one that visits the set of designated states infinitely often.

The product of $\mathcal{A}_{\neg\psi}$ with an MTS $M_P$ that abstracts $P$ is an MTS that retains the type of transitions in $M_P$. We assume that each atomic proposition in the LTL formula is a predicates over the variables, e.g., $x > 4$, $x = y$, etc. The alphabet of $\mathcal{A}_{\neg\psi}$ is then

subsets of these predicates. The transitions in the product of $\mathcal{A}_{\neg\psi}$ and $M_P$ are then such that for two abstract states $a$ and $a'$ of $M_P$ and two states $q$ and $q'$ of $\mathcal{A}_{\neg\psi}$, we have that there is a transition of type $\beta$ (say, $\beta$ is $pmust^+_X$) from the state $\langle a, q\rangle$ of the product to the state $\langle a', q'\rangle$ of the product iff there is a $\beta$-transition from $a$ to $a'$ and there is a transition $\langle q, \sigma, q'\rangle$ of $\mathcal{A}_{\neg\psi}$ such that $a \Rightarrow p$, for all $p \in \sigma$.

Since the product retains the type of transitions of the MTS, the less under-approximating the abstraction is, the more we are likely to detect errors. When $\psi$ is a safety property, $\mathcal{A}_{\neg\psi}$ can be replaced by an automaton accepting finite bad prefixes [KL06], and detection can be reduced to weak reachability in the product. In the general case, we have to find a concrete state that is reachable from itself. The latter cannot be reduced to two weak reachability queries (indeed, the same concrete state has to "glue" the prefix of the lasso with its repeated part, and the same concrete state has to "glue" the repeated parts), but can be reduced to the type of reachability implied by the closure of partitioned-must transitions. Formally, we have the following.

**Theorem 5.** *Consider the MTS $M$ obtained by taking the product of $\mathcal{A}_{\neg\psi}$ and $M_P$. Let $a_{init}$ and $a_{acc}$ be states of $M$ such that $a_{init}$ is initial and $a_{acc}$ is accepting. Consider a set $X \subseteq V$. If $pmust^{+^*}_X(a_{init}, a_{acc})$ and $pmust^{+^*}_X(a_{acc}, a_{acc})$, or $pmust^{-^*}_X(a_{init}, a_{acc})$ and $pmust^{-^*}_X(a_{acc}, a_{acc})$, then $P$ violates $\psi$.*

In Section 5.2, we describe a multi-valued semantics for $\mu$-calculus that is based on partitioned-must transitions and show, for example, that if $pmust^{+^*}_X(a_{init}, a_{acc})$ and $pmust^{+^*}_X(a_{acc}, a_{acc})$, then we can strengthen the conclusion in Theorem 5 to "for every concrete state $c$ that corresponds to $a_{init}$, at least one state in $[c]_X$ violates $\psi$". Nevertheless, the main contribution of partitioned-must transition is not the ability to strengthen the conclusion, but the fact they are applicable in cases usual must transitions fail.

### 5.2 A multi-valued semantics

Since abstraction hides information, the truth value of temporal-logic formulas with respect to states of a MTS may not be definite. According to the three-valued semantics for MTS [GJ02], the value of a formula $\theta$ in abstract state $a$ is **T**, denoted $[a \models \theta] = \mathbf{T}$, only if all the concrete states in $\gamma(a)$ satisfy $\theta$. Likewise, $[a \models \theta] = \mathbf{F}$, only if all the concrete states in $\gamma(a)$ do not satisfy $\theta$. Sometimes, neither case holds, or our reasoning is not sufficiently strong to infer that one of the cases hold [BG00], in which case the value of $\theta$ in $a$ is unknown, denoted $[a \models \theta] = \bot$. Since must transitions under-approximate the transitions of the concrete system, they are used in the three-valued semantics for proving existential properties. Formally, $[a \models \exists\bigcirc\theta] = \mathbf{T}$ iff there is $a'$ such that $must^+(a, a')$ and $[a' \models \theta] = \mathbf{T}$. For logics with backwards modalities, reasoning is the same, with $must^-$ transitions.

By partitioning the variables to relevant ($X$) and irrelevant ones, we can refine the three-valued semantics to one that takes the partition into an account. We say that an abstract state $a$ *$X$-satisfies* a formula $\theta$, denoted $[a \models \theta] \sqsupseteq \mathbf{T}_X$, only if for each state $c \in \gamma(a)$, at least one state in $[c]_X \wedge a$ satisfies $\theta$. Likewise, $a$ does not $X$-satisfy a formula $\theta$, denoted $[a \models \theta] \sqsupseteq \mathbf{F}_X$, if for each state $c \in \gamma(a)$, at least one state in $[c]_X \wedge a$ does not satisfy $\theta$. Note that the conditions for values $\mathbf{T}_{X_1}$ and $\mathbf{T}_{X_2}$, for

$X_1 \neq X_2$, are not mutually exclusive, and so are the conditions for the values $\mathbf{T}_X$ and $\mathbf{F}_X$. This is why we use the $\sqsupseteq$ notation in the definition of the semantics. Formally, the values are taken from the domain $2^V \times 2^V$, where a pair $\langle P, N\rangle \subseteq 2^V \times 2^V$ consists of a *positive set*: all maximal sets $X \subseteq V$ for which $[a \models \theta] \sqsupseteq \mathbf{T}_X$ and a *negative set*: all maximal sets $X \subseteq V$ for which $[a \models \theta] \sqsupseteq \mathbf{F}_X$. Saying that $[a \models \theta] \sqsupseteq \mathbf{T}_X$ means that at least one of the sets in $P$ contains $X$, and similarly for $\mathbf{F}_X$ and $N$. Note that when $P = \emptyset$, the positive set is unknown, and similarly for $N$.

When $X = V$, we have that $[c]_X = \{c\}$. Accordingly, the values $\mathbf{T}_V$ and $\mathbf{F}_V$ coincide with the standard **T** and **F** values from the three-valued semantics. Also, when $X = \emptyset$, we have that $[c]_X = D^V$. Accordingly, the values $\mathbf{T}_\emptyset$ and $\mathbf{F}_\emptyset$ coincide with the existential $\mathbf{T}_\exists$ and $\mathbf{F}_\exists$ values from the six-valued semantics studied in [BKY05]. Finally, it is interesting to note that the semantics is monotonic, in the sense that if $[a \models \theta] \sqsupseteq T_X$ and $X' \subseteq X$, then $[a \models \theta] \sqsupseteq T_{X'}$. Thus, our semantics is a natural refinement of the existential semantics in [BKY05].

As with the existential semantics, however, the weakness of the $\mathbf{T}_X$ and $\mathbf{F}_X$ values is the fact that their conjunction does not correspond to meet in the $(2^V, \subseteq)$ lattice, and results in $\bot$. An exception is the $T_V$ value, where $T_V \wedge T_X = T_X$, for all $X \subseteq V$. Since our main motivation for partitioned-must transitions is reachability, and reachability corresponds to a least fixed point in which the main Boolean operator is a disjunction, the above weakness is not too discouraging. Still, the significance of the semantics here is mainly theoretical, and its goal is to give a logical counterpart of partitioned-must transitions.

Formally, the value of a $\mu$-calculus formula $\theta$ in a state $a$ of a MTS is defined by induction on the structure of $\theta$ as follows. We describe the semantics for *full $\mu$-calculus*, which has both forward ($\exists\bigcirc$) and backwards ($\exists\ominus$) modalities. We assume a $\mu$-calculus in which each atomic proposition is a predicate over the variables, e.g., $x > 4$, $x = y$, etc. We refer to the set of variables appearing in the atomic proposition $p$ by $var(p)$.

$$[a \models p] \sqsupseteq \begin{cases} \mathbf{T}_X & \text{if } (var(p) \subseteq X \text{ and } a \models p) \text{ or} \\ & (var(p) \not\subseteq X \text{ and } [c]_X \wedge a \wedge p \text{ is satisfiable for all } c \in \gamma(a)), \\ \mathbf{F}_X & \text{if } (var(p) \in X \text{ and } a \models \neg p) \text{ or} \\ & (var(p) \notin X \text{ and } [c]_X \wedge a \wedge \neg p \text{ is satisfiable for all } c \in \gamma(a)). \end{cases}$$

$$[a \models \neg\theta] \sqsupseteq \begin{cases} \mathbf{T}_X & \text{if } [a \models \theta] \sqsupseteq \mathbf{F}_X, \\ \mathbf{F}_X & \text{if } [a \models \theta] \sqsupseteq \mathbf{T}_X. \end{cases}$$

$$[a \models \theta \wedge \theta'] \sqsupseteq \begin{cases} \mathbf{T}_X & \text{if } [a \models \theta] \sqsupseteq \mathbf{T}_V \text{ and } [a \models \theta'] \sqsupseteq \mathbf{T}_X. \\ \mathbf{F}_X & \text{if } [a \models \theta] \sqsupseteq \mathbf{F}_V \text{ and } [a \models \theta'] \sqsupseteq \mathbf{F}_X. \end{cases}$$

$$[a \models \theta \vee \theta'] \sqsupseteq \begin{cases} \mathbf{T}_X & \text{if } [a \models \theta] \sqsupseteq \mathbf{T}_X \text{ or } [a \models \theta'] \sqsupseteq \mathbf{T}_X. \\ \mathbf{F}_X & \text{if } [a \models \theta] \sqsupseteq \mathbf{F}_X \text{ or } [a \models \theta'] \sqsupseteq \mathbf{F}_X. \end{cases}$$

$$[a \models \exists\bigcirc\theta] \sqsupseteq \begin{cases} \mathbf{T}_X & \text{if there is } a' \text{ such that } pmust^+_X(a, a') \text{ and } [a' \models_X \theta] \sqsupseteq \mathbf{T}_X, \\ \mathbf{F}_V & \text{if for all } a' \text{ such that } may(a, a'), \text{ we have that } [a' \models_X \theta] \sqsupseteq \mathbf{F}_V. \end{cases}$$

$$[a \models \exists\ominus\theta] \sqsupseteq \begin{cases} \mathbf{T}_X & \text{if there is } a' \text{ such that } pmust^-_X(a', a) \text{ and } [a' \models_X \theta] \sqsupseteq \mathbf{T}_X, \\ \mathbf{F}_V & \text{if there is } a' \text{ such that } may(a', a) \text{ and } [a' \models_X \theta] \sqsupseteq \mathbf{T}_V. \end{cases}$$

Note that when $var(p) \subseteq X$, we have that $[c]_X \wedge a \wedge p$ is satisfiable for all $c \in \gamma(a)$ iff $a \models p$. Thus, the partition to two cases in the base case does not suggest a different semantics for each case and only suggests a simplified check for the case $var(p) \subseteq X$. Note also that for refuting existential properties (equivalently, verifying

universal properties) we proceed with usual may transitions, which corresponds to the case $X = V$. For the fixed-point operators, the closure of partitioned-must transitions under transitivity guarantees we can iterate the local $\exists\bigcirc$ and $\exists\ominus$ modalities, as in the usual three-valued semantics to $\mu$-calculus [BG04]. Note that in the special case of CTL and CTL$^\star$ formulas, this amounts to letting existential path formulas range over $pmust^+_X$ and $pmust^-_X$ paths [SG03].

# 6 Choosing the Relevant Variables

In this section we discuss the choice of the relevant variables. We first show that some of our previous results can be simplified in case the abstraction refers only to variables in $X$. Then, we show that the choice of the relevant variables need not be global, and extend the transitive closure of partitioned-must transitions to cases in which different transitions along the computation require different relevant variables.

## 6.1 An abstraction based on $X$

For a set $X \subseteq V$, we say that an abstraction is *based on* $X$ if all the predicates in $\Phi$ refer only to variables in $X$. When our abstraction is based on $X$, then for all abstract states $a$ and for all $c \in \gamma(a)$, we have $[c]_X \subseteq \gamma(a)$. Accordingly, in the definition of partitioned-must transitions, we can replace $[c]_X \wedge a$ and $[c']_X \wedge a'$ by $[c]_X$ and $[c']_X$, respectively. Consequently, the characterization in Propositions 3 and 4 can be simplified as follows:

**Proposition 5.** *Let $a$ and $a'$ be abstract states in an abstraction that is based on $X$. Then,*

- $pmust^+_X(a, a')$ *only if* $\forall \boldsymbol{x}[a(\boldsymbol{x}) \rightarrow \exists \boldsymbol{x'}.a'(\boldsymbol{x'}) \wedge (\boldsymbol{x'} \rightarrow \mathrm{SP}(s, \boldsymbol{x}))]$.
- $pmust^-_X(a, a')$ *only if* $\forall \boldsymbol{x'}[a'(\boldsymbol{x'}) \rightarrow \exists \boldsymbol{x}.a(\boldsymbol{x'}) \wedge (\boldsymbol{x} \rightarrow \mathrm{WP}(s, \boldsymbol{x'}))]$.

**Example 6** The function gcd described in Example 2 is based in $\{x, y, pc\}$. Hence, the existence of the $pmust^-_{\{x,y,pc\}}$ transitions demonstrated there follows from the validity of the following formulas (since $a$ and $b$ fix $pc$ to 4, we ignore it in the formulas).

- $pmust^-_{\{x,y,pc\}}(a, a)$ iff
  $\forall x', y'[a(x', y') \rightarrow \exists x, y.(a(x, y) \wedge [(x > y \wedge x \neq 2y) \vee (x < y \wedge y \neq 2x)])]$.
- $pmust^-_{\{x,y,pc\}}(a, b)$ iff
  $\forall x', y'[b(x', y') \rightarrow \exists x, y.(a(x, y) \wedge [(x > y \wedge x = 2y) \vee (x < y \wedge y = 2x)])]$.

□

When the specification we want to check involves only predicates that appear in the abstraction, then $var(p) \subseteq X$ for all atomic propositions. Accordingly, for sets $X$ such that the abstraction is based on $X$, the base case of the multi-valued semantics described in Section 5.2 can be simplified, as $[a \models p] \sqsupseteq \mathbf{T}_X$ if $a \models p$ and $[a \models p] \sqsupseteq \mathbf{F}_X$ if $a \models \neg p$.

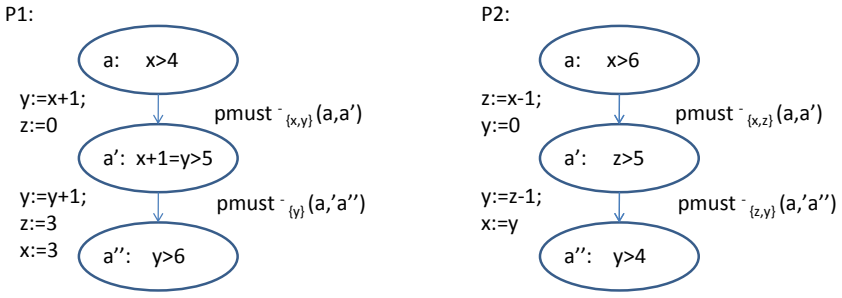


**Fig. 4.** Existence and nonexistence of dynamic transitive closure.

### 6.2 Choosing $X$

Recall that our motivation is to detect weak reachability in the concrete system. Proposition 1 shows that partitioned-must transitions are closed under transitivity and can therefore be used for showing weak reachability. The theorem, however, assumes one set $X$ with respect to which we partition all the transitions along the path. Below we generalize the proposition to a dynamic choice of sets according to which the transitions are partitioned.

**Proposition 6. [dynamic transitive closure]** *Let $a_1, a_2, \ldots, a_n$ be a sequence of abstract states.*

1. *If there is a sequence $X_1 \subseteq X_2 \subseteq \cdots \subseteq X_{n-1} \subseteq V$ such that for all $1 \leq i < n$, we have that $pmust^+_{X_i}(a_i, a_{i+1})$, then for every concrete state $c_1 \in \gamma(a_1)$, there is a concrete state $c_n \in \gamma(a_n)$, such that $must^{-*}([c_1]_{X_1} \wedge a_1, [c_n]_{X_{n-1}} \wedge a_n)$.*
2. *If there is a sequence $V \supseteq X_1 \supseteq X_2 \cdots \subseteq X_{n-1}$ such that for all $1 \leq i < n$, we have that $pmust^-_{X_i}(a_i, a_{i+1})$, then for every concrete state $c_n \in \gamma(a_n)$, there is a concrete state $c_1 \in \gamma(a_1)$, such that $must^{-*}([c_1]_{X_1} \wedge a_1, [c_n]_{X_{n-1}} \wedge a_n)$.*

The proof of the proposition proceeds by an induction on $n$ and is similar to the proof of Proposition 1.

*Remark 2.* It is shown in [Bal04] that weak reachability in a framework with no partitioned-must transitions follows from a sequence of $must^-$ transitions followed by a sequence of $must^+$ transitions. By Proposition 2, $must^+(a, a')$ iff $pmust^+_V(a, a')$ and $must^-(a, a')$ iff $pmust^+_\emptyset(a, a')$. Likewise, since $must^+(a, a')$ iff $pmust^-_\emptyset(a, a')$ and $must^-(a, a')$ iff $pmust^-_V(a, a')$, it is also a special case of the dynamic transitive closure of $pmust^-$ transitions.

**Example 7** Consider the program $P_1$ appearing in Figure 4. Since $pmust^-_{\{x,y\}}(a, a')$ and $pmust^-_{\{y\}}(a', a'')$, we can conclude that for every concrete state $c'$ satisfying $y > 6$ there is a concrete state $c$ satisfying $x > 4$ such that all the states satisfying $x > 4$ and that agree with $c$ on the values of $x$ and $y$ can reach states that satisfy $y > 6$ and agree with $c'$ on the value of $y$. Indeed, if $c' = \langle x, y, z\rangle$, then we can take $c = \langle y - 1, 0, 0\rangle$. Note that nor $pmust^-_{\{y\}}(a, a')$ neither $pmust^-_{\{x,y\}}(a', a'')$. Thus, the dynamic choice of relevant variables is essential.

Consider now the program $P_2$ in the figure. It requires a dynamic choice of relevant variables that does not satisfy the conditions of Proposition 6, as $\{z, y\} \not\subseteq \{x, y\}$. This is unfortunate, as it is true that for every concrete state $c'$ satisfying $y > 4$ there is a concrete state $c$ satisfying $x > 6$ such that $must^+([c]_{\{x\}}, [c']_{\{y\}})$. The cause of this inapplicability of Proposition 6 is the fact that the assignments in the program correspond to renaming of the variables. To see this, consider a program $P_2'$ in which the only variables are $x$ and $y$, the abstract states are $(x > 6)$, $(x > 5)$, and $(x > 4)$, and statements are obtained from these of $P_2$ by renaming $z$ to $x$ in the first and second transitions and renaming $y$ to $x$ and $x$ to $y$ in the second transition. Then, we can use $pmust^-_{\{x\}}$-transitions in order to prove that for every concrete state $c' \in \gamma(a'')$, there is concrete state $c \in \gamma(a)$ such that $must^+([c]_{\{x\}}, [c']_{\{x\}})$. □

## References


[Bal04] T. Ball. A theory of predicate-complete test coverage and generation. In *3rd International Symposium on Formal Methods for Components and Objects*, 2004.

[BG00] G. Bruns and P. Godefroid. Generalized model checking: Reasoning about partial state spaces. In *Proc. 11th International Conference on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 168–182, 2000.

[BG04] G. Bruns and P. Godefroid. Model checking with 3-valued temporal logics. In *31st International Colloquium on Automata, Languages and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 281–293, 2004.

[BKY05] T. Ball, O. Kupferman, and G. Yorsh. Abstraction for falsification. In *Proc. 17th CAV*, LNCS 3578, pages 67–81, 2005.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for the static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM POPL*, pages 238–252, 1977.

[Dij76] E.W. Dijksta. *A Discipline of Programming*. Prentice-Hall, 1976.

[GJ02] P. Godefroid and R. Jagadeesan. Automatic abstraction using generalized model checking. In *Proc. 14th CAV*, LNCS 2404, pages 137–150, 2002.

[KL06] O. Kupferman and R. Lampert. On the construction of fine automata for safety properties. In *Proc. 4th ATVA*, LNCS 4218, pages 110–124, 2006.

[Kur94] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.

[LT88] K.G. Larsen and G.B. Thomsen. A modal process logic. In *Proc. 3th LICS*, 1988.

[LX90] K.G. Larsen and L. XinXin. Equation solving using modal transition systems. In *Proc. 5th LICS*, pages 108–117, Philadelphia, June 1990.

[SG03] S. Shoham and O. Grumberg. A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. In *Proc. 15th CAV*, LNCS 2725, pages 275–287, 2003.

[SG04] S. Shoham and O. Grumberg. Monotonic abstraction-refinement for CTL. In *Proc. 10th TACAS*, LNCS 2988, pages 546–560, 2004.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *I & C*, 115(1):1–37, November 1994.