Program Verification: Lecture 18

José Meseguer

University of Illinois at Urbana-Champaign

The Mathematical Model of a Rewrite Theory

I have been stating all along that Maude Programming is Mathematical Modeling,

The Mathematical Model of a Rewrite Theory

I have been stating all along that Maude Programming is Mathematical Modeling, and that:

The Mathematical Model of a Rewrite Theory

I have been stating all along that Maude Programming is Mathematical Modeling, and that:

The meaning of a Maude program P is a mathematical model $\mathbb{C}_P$, called its canonical model.

The Mathematical Model of a Rewrite Theory

I have been stating all along that Maude Programming is Mathematical Modeling, and that:

The meaning of a Maude program P is a mathematical model $\mathbb{C}_P$, called its canonical model.

For a functional module fmod $(\Sigma, E \cup B)$ endfm that is (1) ground convergent and sufficiently complete for constructors $\Omega \subseteq \Sigma$, the canonical model is the canonical term algebra $\mathbb{C}_{\Sigma/\vec{E},B}$.

The Mathematical Model of a Rewrite Theory

I have been stating all along that Maude Programming is Mathematical Modeling, and that:

The meaning of a Maude program P is a mathematical model $\mathbb{C}_P$, called its canonical model.

For a functional module `fmod` $(\Sigma, E \cup B)$ `endfm` that is (1) ground convergent and sufficiently complete for constructors $\Omega \subseteq \Sigma$, the canonical model is the canonical term algebra $\mathbb{C}_{\Sigma/\vec{E},B}$. But what is the model $\mathbb{C}_P$ for P a system module `mod` $(\Sigma, E \cup B, R)$ `endm`?

The Mathematical Model of a Rewrite Theory

I have been stating all along that Maude Programming is Mathematical Modeling, and that:

The meaning of a Maude program P is a mathematical model $\mathbb{C}_P$, called its canonical model.

For a functional module `fmod` $(\Sigma, E \cup B)$ `endfm` that is (1) ground convergent and sufficiently complete for constructors $\Omega \subseteq \Sigma$, the canonical model is the canonical term algebra $\mathbb{C}_{\Sigma/\vec{E},B}$. But what is the model $\mathbb{C}_P$ for P a system module `mod` $(\Sigma, E \cup B, R)$ `endm`?

Intuitively, it should be a transition system. Its functional part $(\Sigma, E \cup B)$ should be an equational theory satisfying requirement (1) .

The Mathematical Model of a Rewrite Theory

I have been stating all along that Maude Programming is Mathematical Modeling, and that:

The meaning of a Maude program P is a mathematical model $\mathbb{C}_P$, called its canonical model.

For a functional module `fmod` $(\Sigma, E \cup B)$ `endfm` that is (1) ground convergent and sufficiently complete for constructors $\Omega \subseteq \Sigma$, the canonical model is the canonical term algebra $\mathbb{C}_{\Sigma/\vec{E},B}$. But what is the model $\mathbb{C}_P$ for P a system module `mod` $(\Sigma, E \cup B, R)$ `endm`?

Intuitively, it should be a transition system. Its functional part $(\Sigma, E \cup B)$ should be an equational theory satisfying requirement (1) . Therefore its states should the the elements of the canonical term algebra $\mathbb{C}_{\Sigma/\vec{E},B}$.

The Mathematical Model of a Rewrite Theory

I have been stating all along that Maude Programming is Mathematical Modeling, and that:

The meaning of a Maude program P is a mathematical model $\mathbb{C}_P$, called its canonical model.

For a functional module `fmod` $(\Sigma, E \cup B)$ `endfm` that is (1) ground convergent and sufficiently complete for constructors $\Omega \subseteq \Sigma$, the canonical model is the canonical term algebra $\mathbb{C}_{\Sigma/\vec{E},B}$. But what is the model $\mathbb{C}_P$ for P a system module `mod` $(\Sigma, E \cup B, R)$ `endm`?

Intuitively, it should be a transition system. Its functional part $(\Sigma, E \cup B)$ should be an equational theory satisfying requirement (1) . Therefore its states should the the elements of the canonical term algebra $\mathbb{C}_{\Sigma/\vec{E},B}$. What about its transition relation?

The Mathematical Model of a Rewrite Theory

I have been stating all along that Maude Programming is Mathematical Modeling, and that:

The meaning of a Maude program P is a mathematical model $\mathbb{C}_P$, called its canonical model.

For a functional module `fmod` $(\Sigma, E \cup B)$ `endfm` that is (1) ground convergent and sufficiently complete for constructors $\Omega \subseteq \Sigma$, the canonical model is the canonical term algebra $\mathbb{C}_{\Sigma/\vec{E},B}$. But what is the model $\mathbb{C}_P$ for P a system module `mod` $(\Sigma, E \cup B, R)$ `endm`?

Intuitively, it should be a transition system. Its functional part $(\Sigma, E \cup B)$ should be an equational theory satisfying requirement (1) . Therefore its states should the the elements of the canonical term algebra $\mathbb{C}_{\Sigma/\vec{E},B}$. What about its transition relation? They should be transitions defined by the rules R.

The Mathematical Model of a Rewrite Theory (II)

But there is a problem, called the coherence problem.

The Mathematical Model of a Rewrite Theory (II)

But there is a problem, called the coherence problem. Let (Σ, E, R) have Σ unsorted with just three constants a, b, c, $E = \{a = c\}$, and $R = \{a \to b\}$, with $\Omega = \{c, b\}$, so that $\mathbb{C}_{\Sigma/E} = \{c, b\}$ has just two states.

The Mathematical Model of a Rewrite Theory (II)

But there is a problem, called the coherence problem. Let (Σ, E, R) have Σ unsorted with just three constants a, b, c, $E = \{a = c\}$, and $R = \{a \to b\}$, with $\Omega = \{c, b\}$, so that $\mathbb{C}_{\Sigma/E} = \{c, b\}$ has just two states. The problem is that there is no meaningful way to apply the rule $a \to b$ to obtain the transition that should exist from state c to state b.

The Mathematical Model of a Rewrite Theory (II)

But there is a problem, called the coherence problem. Let (Σ, E, R) have Σ unsorted with just three constants a, b, c, $E = \{a = c\}$, and $R = \{a \to b\}$, with $\Omega = \{c, b\}$, so that $\mathbb{C}_{\Sigma/E} = \{c, b\}$ has just two states. The problem is that there is no meaningful way to apply the rule $a \to b$ to obtain the transition that should exist from state c to state b.

The mathematical model we want is called a Σ-transition system,

The Mathematical Model of a Rewrite Theory (II)

But there is a problem, called the coherence problem. Let (Σ, E, R) have Σ unsorted with just three constants a, b, c, $E = \{a = c\}$, and $R = \{a \to b\}$, with $\Omega = \{c, b\}$, so that $\mathbb{C}_{\Sigma/E} = \{c, b\}$ has just two states. The problem is that there is no meaningful way to apply the rule $a \to b$ to obtain the transition that should exist from state c to state b.

The mathematical model we want is called a Σ-transition system, where the states have a Σ-algebra structure

The Mathematical Model of a Rewrite Theory (II)

But there is a problem, called the coherence problem. Let (Σ, E, R) have Σ unsorted with just three constants a, b, c, $E = \{a = c\}$, and $R = \{a \to b\}$, with $\Omega = \{c, b\}$, so that $\mathbb{C}_{\Sigma/E} = \{c, b\}$ has just two states. The problem is that there is no meaningful way to apply the rule $a \to b$ to obtain the transition that should exist from state c to state b.

The mathematical model we want is called a Σ-transition system, where the states have a Σ-algebra structure —in our case $\mathbb{C}_{\Sigma/\vec{E},B}$—

The Mathematical Model of a Rewrite Theory (II)

But there is a problem, called the coherence problem. Let (Σ, E, R) have Σ unsorted with just three constants a, b, c, $E = \{a = c\}$, and $R = \{a \to b\}$, with $\Omega = \{c, b\}$, so that $\mathbb{C}_{\Sigma/E} = \{c, b\}$ has just two states. The problem is that there is no meaningful way to apply the rule $a \to b$ to obtain the transition that should exist from state c to state b.

The mathematical model we want is called a Σ-transition system, where the states have a Σ-algebra structure —in our case $\mathbb{C}_{\Sigma/\vec{E},B}$— and there is a transition relation between states.

The Mathematical Model of a Rewrite Theory (II)

But there is a problem, called the coherence problem. Let (Σ, E, R) have Σ unsorted with just three constants a, b, c, $E = \{a = c\}$, and $R = \{a \to b\}$, with $\Omega = \{c, b\}$, so that $\mathbb{C}_{\Sigma/E} = \{c, b\}$ has just two states. The problem is that there is no meaningful way to apply the rule $a \to b$ to obtain the transition that should exist from state c to state b.

The mathematical model we want is called a Σ-transition system, where the states have a Σ-algebra structure —in our case $\mathbb{C}_{\Sigma/\vec{E},B}$— and there is a transition relation between states. We just need to have a suitable executability requirement

The Mathematical Model of a Rewrite Theory (II)

But there is a problem, called the coherence problem. Let (Σ, E, R) have Σ unsorted with just three constants a, b, c, $E = \{a = c\}$, and $R = \{a \to b\}$, with $\Omega = \{c, b\}$, so that $\mathbb{C}_{\Sigma/E} = \{c, b\}$ has just two states. The problem is that there is no meaningful way to apply the rule $a \to b$ to obtain the transition that should exist from state c to state b.

The mathematical model we want is called a Σ-transition system, where the states have a Σ-algebra structure —in our case $\mathbb{C}_{\Sigma/\vec{E},B}$— and there is a transition relation between states. We just need to have a suitable executability requirement (besides requirement (1) for the equations)

The Mathematical Model of a Rewrite Theory (II)

But there is a problem, called the coherence problem. Let (Σ, E, R) have Σ unsorted with just three constants a, b, c, $E = \{a = c\}$, and $R = \{a \to b\}$, with $\Omega = \{c, b\}$, so that $\mathbb{C}_{\Sigma/E} = \{c, b\}$ has just two states. The problem is that there is no meaningful way to apply the rule $a \to b$ to obtain the transition that should exist from state c to state b.

The mathematical model we want is called a Σ-transition system, where the states have a Σ-algebra structure —in our case $\mathbb{C}_{\Sigma/\vec{E},B}$— and there is a transition relation between states. We just need to have a suitable executability requirement (besides requirement (1) for the equations) to properly define the state transition relation in our desired canonical Σ-transition system.

Executability of Rewrite Theories: Coherence

When is a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ executable?

Executability of Rewrite Theories: Coherence

When is a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ executable?
$(\Sigma, E \cup B)$ with constructors Ω should satisfy the requirement (1) of functional modules.

Executability of Rewrite Theories: Coherence

When is a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ executable? $(\Sigma, E \cup B)$ with constructors Ω should satisfy the requirement (1) of functional modules. But this is not enough.

Executability of Rewrite Theories: Coherence

When is a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ executable?
$(\Sigma, E \cup B)$ with constructors Ω should satisfy the requirement (1) of functional modules. But this is not enough. We also need that:

Executability of Rewrite Theories: Coherence

When is a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ executable?

$(\Sigma, E \cup B)$ with constructors Ω should satisfy the requirement (1) of functional modules. But this is not enough. We also need that:

(2) the rules R are ground coherent with E modulo the axioms B

Executability of Rewrite Theories: Coherence

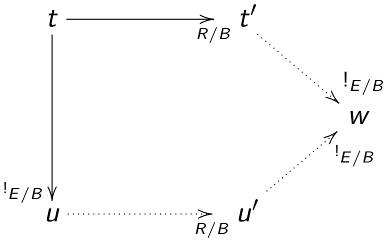
When is a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ executable?
$(\Sigma, E \cup B)$ with constructors Ω should satisfy the requirement (1) of functional modules. But this is not enough. We also need that:
(2) the rules R are ground coherent with E modulo the axioms B (in our example, we just need to add rule $c \to b$).

Executability of Rewrite Theories: Coherence

When is a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ executable?
$(\Sigma, E \cup B)$ with constructors Ω should satisfy the requirement (1) of functional modules. But this is not enough. We also need that:
(2) the rules R are ground coherent with E modulo the axioms B (in our example, we just need to add rule $c \rightarrow b$). This requirement is captured by the diagram (dotted arrows existential):

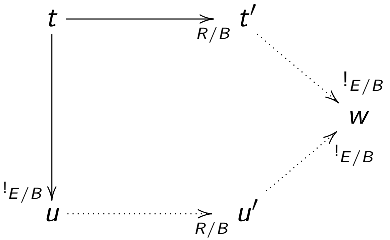
Executability of Rewrite Theories: Coherence

When is a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ executable? $(\Sigma, E \cup B)$ with constructors Ω should satisfy the requirement (1) of functional modules. But this is not enough. We also need that: (2) the rules R are ground coherent with E modulo the axioms B (in our example, we just need to add rule $c \to b$). This requirement is captured by the diagram (dotted arrows existential):



Executability of Rewrite Theories: Coherence

When is a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ executable? $(\Sigma, E \cup B)$ with constructors Ω should satisfy the requirement (1) of functional modules. But this is not enough. We also need that: (2) the rules R are ground coherent with E modulo the axioms B (in our example, we just need to add rule $c \rightarrow b$). This requirement is captured by the diagram (dotted arrows existential):



Maude's Coherence Checker tool checks this property.

The Canonical Σ-Transition System $\mathbb{C}_{\mathcal{R}}$

Given a system module `mod` $\mathcal{R}$ `endm`, with, say, $\mathcal{R} = (\Sigma, E \cup B, R)$, Maude assumes executability requirement (1) for $(\Sigma, E \cup B)$,

The Canonical Σ-Transition System $\mathbb{C}_{\mathcal{R}}$

Given a system module `mod` $\mathcal{R}$ `endm`, with, say, $\mathcal{R} = (\Sigma, E \cup B, R)$, Maude assumes executability requirement (1) for $(\Sigma, E \cup B)$, and (2) ground coherence of R w.r.t. E modulo B.

The Canonical Σ-Transition System $\mathbb{C}_{\mathcal{R}}$

Given a system module `mod` $\mathcal{R}$ `endm`, with, say, $\mathcal{R} = (\Sigma, E \cup B, R)$, Maude assumes executability requirement (1) for $(\Sigma, E \cup B)$, and (2) ground coherence of R w.r.t. E modulo B.

Assuming (1)–(2), the mathematical model of `mod` $\mathcal{R}$ `endm` is the canonical Σ-transition system $\mathbb{C}_{\mathcal{R}} = (\mathbb{C}_{\Sigma/\vec{E},B}, \rightarrow_{\mathbb{C}_{\mathcal{R}}})$,

The Canonical Σ-Transition System $\mathbb{C}_{\mathcal{R}}$

Given a system module `mod` $\mathcal{R}$ `endm`, with, say, $\mathcal{R} = (\Sigma, E \cup B, R)$, Maude assumes executability requirement (1) for $(\Sigma, E \cup B)$, and (2) ground coherence of R w.r.t. E modulo B.

Assuming (1)–(2), the mathematical model of `mod` $\mathcal{R}$ `endm` is the canonical Σ-transition system $\mathbb{C}_{\mathcal{R}} = (\mathbb{C}_{\Sigma/\vec{E},B}, \rightarrow_{\mathbb{C}_{\mathcal{R}}})$, were $\mathbb{C}_{\Sigma/\vec{E},B}$ is the canonical term algebra;

The Canonical Σ-Transition System $\mathbb{C}_{\mathcal{R}}$

Given a system module `mod` $\mathcal{R}$ `endm`, with, say, $\mathcal{R} = (\Sigma, E \cup B, R)$, Maude assumes executability requirement (1) for $(\Sigma, E \cup B)$, and (2) ground coherence of R w.r.t. E modulo B.

Assuming (1)–(2), the mathematical model of `mod` $\mathcal{R}$ `endm` is the canonical Σ-transition system $\mathbb{C}_{\mathcal{R}} = (\mathbb{C}_{\Sigma/\vec{E},B}, \to_{\mathbb{C}_{\mathcal{R}}})$, were $\mathbb{C}_{\Sigma/\vec{E},B}$ is the canonical term algebra; and given $[u], [v] \in C_{\Sigma/\vec{E},B,[s]}$, $[u] \to_{\mathbb{C}_{\mathcal{R}}} [v]$ holds iff

The Canonical Σ-Transition System $\mathbb{C}_{\mathcal{R}}$

Given a system module `mod` $\mathcal{R}$ `endm`, with, say, $\mathcal{R} = (\Sigma, E \cup B, R)$, Maude assumes executability requirement (1) for $(\Sigma, E \cup B)$, and (2) ground coherence of R w.r.t. E modulo B.

Assuming (1)–(2), the mathematical model of `mod` $\mathcal{R}$ `endm` is the canonical Σ-transition system $\mathbb{C}_{\mathcal{R}} = (\mathbb{C}_{\Sigma/\vec{E},B}, \rightarrow_{\mathbb{C}_{\mathcal{R}}})$, were $\mathbb{C}_{\Sigma/\vec{E},B}$ is the canonical term algebra; and given $[u], [v] \in C_{\Sigma/\vec{E},B,[s]}$, $[u] \rightarrow_{\mathbb{C}_{\mathcal{R}}} [v]$ holds iff there exists v' such that $u \rightarrow_{R/B} v'$ and $[v] = [v'!_{E/B}]$.

The Canonical Σ-Transition System $\mathbb{C}_{\mathcal{R}}$

Given a system module `mod` $\mathcal{R}$ `endm`, with, say, $\mathcal{R} = (\Sigma, E \cup B, R)$, Maude assumes executability requirement (1) for $(\Sigma, E \cup B)$, and (2) ground coherence of R w.r.t. E modulo B.

Assuming (1)–(2), the mathematical model of `mod` $\mathcal{R}$ `endm` is the canonical Σ-transition system $\mathbb{C}_{\mathcal{R}} = (\mathbb{C}_{\Sigma/\vec{E},B}, \to_{\mathbb{C}_{\mathcal{R}}})$, were $\mathbb{C}_{\Sigma/\vec{E},B}$ is the canonical term algebra; and given $[u], [v] \in C_{\Sigma/\vec{E},B,[s]}$, $[u] \to_{\mathbb{C}_{\mathcal{R}}} [v]$ holds iff there exists v' such that $u \to_{R/B} v'$ and $[v] = [v'!_{E/B}]$. I.e., states are elements of $\mathbb{C}_{\Sigma/E,B}$; and

The Canonical Σ-Transition System $\mathbb{C}_{\mathcal{R}}$

Given a system module `mod` $\mathcal{R}$ `endm`, with, say, $\mathcal{R} = (\Sigma, E \cup B, R)$, Maude assumes executability requirement (1) for $(\Sigma, E \cup B)$, and (2) ground coherence of R w.r.t. E modulo B.

Assuming (1)–(2), the mathematical model of `mod` $\mathcal{R}$ `endm` is the canonical Σ-transition system $\mathbb{C}_{\mathcal{R}} = (\mathbb{C}_{\Sigma/\vec{E},B}, \rightarrow_{\mathbb{C}_{\mathcal{R}}})$, were $\mathbb{C}_{\Sigma/\vec{E},B}$ is the canonical term algebra; and given $[u], [v] \in C_{\Sigma/\vec{E},B,[s]}$, $[u] \rightarrow_{\mathbb{C}_{\mathcal{R}}} [v]$ holds iff there exists v' such that $u \rightarrow_{R/B} v'$ and $[v] = [v'!_{E/B}]$. I.e., states are elements of $\mathbb{C}_{\Sigma/E,B}$; and transitions from $[u] \in \mathbb{C}_{\Sigma/E,B}$,

The Canonical Σ-Transition System $\mathbb{C}_{\mathcal{R}}$

Given a system module `mod` $\mathcal{R}$ `endm`, with, say, $\mathcal{R} = (\Sigma, E \cup B, R)$, Maude assumes executability requirement (1) for $(\Sigma, E \cup B)$, and (2) ground coherence of R w.r.t. E modulo B.

Assuming (1)–(2), the mathematical model of `mod` $\mathcal{R}$ `endm` is the canonical Σ-transition system $\mathbb{C}_{\mathcal{R}} = (\mathbb{C}_{\Sigma/\vec{E},B}, \to_{\mathbb{C}_{\mathcal{R}}})$, were $\mathbb{C}_{\Sigma/\vec{E},B}$ is the canonical term algebra; and given $[u], [v] \in C_{\Sigma/\vec{E},B,[s]}$, $[u] \to_{\mathbb{C}_{\mathcal{R}}} [v]$ holds iff there exists v' such that $u \to_{R/B} v'$ and $[v] = [v'!_{E/B}]$. I.e., states are elements of $\mathbb{C}_{\Sigma/E,B}$; and transitions from $[u] \in \mathbb{C}_{\Sigma/E,B}$, denoted $[u] \to_{\mathbb{C}_{\mathcal{R}}} [v]$,

The Canonical Σ-Transition System $\mathbb{C}_{\mathcal{R}}$

Given a system module `mod` $\mathcal{R}$ `endm`, with, say, $\mathcal{R} = (\Sigma, E \cup B, R)$, Maude assumes executability requirement (1) for $(\Sigma, E \cup B)$, and (2) ground coherence of R w.r.t. E modulo B.

Assuming (1)–(2), the mathematical model of `mod` $\mathcal{R}$ `endm` is the canonical Σ-transition system $\mathbb{C}_{\mathcal{R}} = (\mathbb{C}_{\Sigma/\vec{E},B}, \rightarrow_{\mathbb{C}_{\mathcal{R}}})$, were $\mathbb{C}_{\Sigma/\vec{E},B}$ is the canonical term algebra; and given $[u], [v] \in C_{\Sigma/\vec{E},B,[s]}$, $[u] \rightarrow_{\mathbb{C}_{\mathcal{R}}} [v]$ holds iff there exists v' such that $u \rightarrow_{R/B} v'$ and $[v] = [v'!_{E/B}]$. I.e., states are elements of $\mathbb{C}_{\Sigma/E,B}$; and transitions from $[u] \in \mathbb{C}_{\Sigma/E,B}$, denoted $[u] \rightarrow_{\mathbb{C}_{\mathcal{R}}} [v]$, are those such that there exists a one-step rewrite $u \rightarrow_{R/B} v'$ s.t. $[v] = [v'!_{E/B}]$.

The Canonical Σ-Transition System $\mathbb{C}_{\mathcal{R}}$

Given a system module `mod` $\mathcal{R}$ `endm`, with, say, $\mathcal{R} = (\Sigma, E \cup B, R)$, Maude assumes executability requirement (1) for $(\Sigma, E \cup B)$, and (2) ground coherence of R w.r.t. E modulo B.

Assuming (1)–(2), the mathematical model of `mod` $\mathcal{R}$ `endm` is the canonical Σ-transition system $\mathbb{C}_{\mathcal{R}} = (\mathbb{C}_{\Sigma/\vec{E},B}, \rightarrow_{\mathbb{C}_{\mathcal{R}}})$, were $\mathbb{C}_{\Sigma/\vec{E},B}$ is the canonical term algebra; and given $[u],[v] \in C_{\Sigma/\vec{E},B,[s]}$, $[u] \rightarrow_{\mathbb{C}_{\mathcal{R}}} [v]$ holds iff there exists v' such that $u \rightarrow_{R/B} v'$ and $[v] = [v'!_{E/B}]$. I.e., states are elements of $\mathbb{C}_{\Sigma/E,B}$; and transitions from $[u] \in \mathbb{C}_{\Sigma/E,B}$, denoted $[u] \rightarrow_{\mathbb{C}_{\mathcal{R}}} [v]$, are those such that there exists a one-step rewrite $u \rightarrow_{R/B} v'$ s.t. $[v] = [v'!_{E/B}]$.

That is, the states reachable from state $[u]$ by a $\rightarrow_{\mathbb{C}_{\mathcal{R}}}$-transition

The Canonical Σ-Transition System $\mathbb{C}_{\mathcal{R}}$

Given a system module `mod` $\mathcal{R}$ `endm`, with, say, $\mathcal{R} = (\Sigma, E \cup B, R)$, Maude assumes executability requirement (1) for $(\Sigma, E \cup B)$, and (2) ground coherence of R w.r.t. E modulo B.

Assuming (1)–(2), the mathematical model of `mod` $\mathcal{R}$ `endm` is the canonical Σ-transition system $\mathbb{C}_{\mathcal{R}} = (\mathbb{C}_{\Sigma/\vec{E},B}, \rightarrow_{\mathbb{C}_{\mathcal{R}}})$, were $\mathbb{C}_{\Sigma/\vec{E},B}$ is the canonical term algebra; and given $[u], [v] \in C_{\Sigma/\vec{E},B,[s]}$, $[u] \rightarrow_{\mathbb{C}_{\mathcal{R}}} [v]$ holds iff there exists v' such that $u \rightarrow_{R/B} v'$ and $[v] = [v'!_{E/B}]$. I.e., states are elements of $\mathbb{C}_{\Sigma/E,B}$; and transitions from $[u] \in \mathbb{C}_{\Sigma/E,B}$, denoted $[u] \rightarrow_{\mathbb{C}_{\mathcal{R}}} [v]$, are those such that there exists a one-step rewrite $u \rightarrow_{R/B} v'$ s.t. $[v] = [v'!_{E/B}]$.

That is, the states reachable from state $[u]$ by a $\rightarrow_{\mathbb{C}_{\mathcal{R}}}$-transition are the normal forms of its 1-step $\rightarrow_{R/B}$-rewrites from $[u]$.

Specifying and Verifying Properties of Concurrent Systems

For a concurrent system specified by a rewrite theory $\mathcal{R}$ enjoying properties (1)–(2), what does it mean to assert that it satisfies some formal property φ?

Specifying and Verifying Properties of Concurrent Systems

For a concurrent system specified by a rewrite theory $\mathcal{R}$ enjoying properties (1)–(2), what does it mean to assert that it satisfies some formal property φ? It should exactly mean that $\mathbb{C}_{\mathcal{R}} \models \varphi$.

Specifying and Verifying Properties of Concurrent Systems

For a concurrent system specified by a rewrite theory $\mathcal{R}$ enjoying properties (1)–(2), what does it mean to assert that it satisfies some formal property φ? It should exactly mean that $\mathbb{C}_{\mathcal{R}} \models \varphi$. The property φ in question can be specified in some property specification logic of our choice such as, for example,

Specifying and Verifying Properties of Concurrent Systems

For a concurrent system specified by a rewrite theory $\mathcal{R}$ enjoying properties (1)–(2), what does it mean to assert that it satisfies some formal property φ? It should exactly mean that $\mathbb{C}_{\mathcal{R}} \models \varphi$. The property φ in question can be specified in some property specification logic of our choice such as, for example,

1. Modal Logic

Specifying and Verifying Properties of Concurrent Systems

For a concurrent system specified by a rewrite theory $\mathcal{R}$ enjoying properties (1)–(2), what does it mean to assert that it satisfies some formal property φ? It should exactly mean that $\mathbb{C}_{\mathcal{R}} \models \varphi$. The property φ in question can be specified in some property specification logic of our choice such as, for example,

1. Modal Logic
2. Temporal Logic

Specifying and Verifying Properties of Concurrent Systems

For a concurrent system specified by a rewrite theory $\mathcal{R}$ enjoying properties (1)–(2), what does it mean to assert that it satisfies some formal property φ? It should exactly mean that $\mathbb{C}_{\mathcal{R}} \models \varphi$. The property φ in question can be specified in some property specification logic of our choice such as, for example,

1. Modal Logic
2. Temporal Logic
3. Reachability Logic (which includes Hoare Logic)

Specifying and Verifying Properties of Concurrent Systems

For a concurrent system specified by a rewrite theory $\mathcal{R}$ enjoying properties (1)–(2), what does it mean to assert that it satisfies some formal property φ? It should exactly mean that $\mathbb{C}_{\mathcal{R}} \models \varphi$. The property φ in question can be specified in some property specification logic of our choice such as, for example,

1. Modal Logic
2. Temporal Logic
3. Reachability Logic (which includes Hoare Logic)

Maude tools can be used to verify that $\mathbb{C}_{\mathcal{R}} \models \varphi$ in logics (1)–(3).

Specifying and Verifying Properties of Concurrent Systems

For a concurrent system specified by a rewrite theory $\mathcal{R}$ enjoying properties (1)–(2), what does it mean to assert that it satisfies some formal property φ? It should exactly mean that $\mathbb{C}_{\mathcal{R}} \models \varphi$. The property φ in question can be specified in some property specification logic of our choice such as, for example,

1. Modal Logic
2. Temporal Logic
3. Reachability Logic (which includes Hoare Logic)

Maude tools can be used to verify that $\mathbb{C}_{\mathcal{R}} \models \varphi$ in logics (1)–(3).

In this course we shall verify properties of $\mathbb{C}_{\mathcal{R}}$ in both modal logic and (linear time) temporal logic (LTL) by:

Specifying and Verifying Properties of Concurrent Systems

For a concurrent system specified by a rewrite theory $\mathcal{R}$ enjoying properties (1)–(2), what does it mean to assert that it satisfies some formal property φ? It should exactly mean that $\mathbb{C}_{\mathcal{R}} \models \varphi$. The property φ in question can be specified in some property specification logic of our choice such as, for example,

1. Modal Logic
2. Temporal Logic
3. Reachability Logic (which includes Hoare Logic)

Maude tools can be used to verify that $\mathbb{C}_{\mathcal{R}} \models \varphi$ in logics (1)–(3).

In this course we shall verify properties of $\mathbb{C}_{\mathcal{R}}$ in both modal logic and (linear time) temporal logic (LTL) by:

- Explicit-state model checking.

Specifying and Verifying Properties of Concurrent Systems

For a concurrent system specified by a rewrite theory $\mathcal{R}$ enjoying properties (1)–(2), what does it mean to assert that it satisfies some formal property φ? It should exactly mean that $\mathbb{C}_{\mathcal{R}} \models \varphi$. The property φ in question can be specified in some property specification logic of our choice such as, for example,

1. Modal Logic
2. Temporal Logic
3. Reachability Logic (which includes Hoare Logic)

Maude tools can be used to verify that $\mathbb{C}_{\mathcal{R}} \models \varphi$ in logics (1)–(3).

In this course we shall verify properties of $\mathbb{C}_{\mathcal{R}}$ in both modal logic and (linear time) temporal logic (LTL) by:

- Explicit-state model checking.
- Symbolic model checking.

Specifying and Verifying Properties of Concurrent Systems

For a concurrent system specified by a rewrite theory $\mathcal{R}$ enjoying properties (1)–(2), what does it mean to assert that it satisfies some formal property φ? It should exactly mean that $\mathbb{C}_{\mathcal{R}} \models \varphi$.

The property φ in question can be specified in some property specification logic of our choice such as, for example,

1. Modal Logic
2. Temporal Logic
3. Reachability Logic (which includes Hoare Logic)

Maude tools can be used to verify that $\mathbb{C}_{\mathcal{R}} \models \varphi$ in logics (1)–(3).

In this course we shall verify properties of $\mathbb{C}_{\mathcal{R}}$ in both modal logic and (linear time) temporal logic (LTL) by:

- Explicit-state model checking.
- Symbolic model checking.
- Symbolic model checking + Theorem proving.

Modal Logic

Modal logic is a logic to reason about necessity and possibility of future events that goes back to Aristotle.

Modal Logic

Modal logic is a logic to reason about necessity and possibility of future events that goes back to Aristotle. As an example of modal logic reasoning, Aristotle poses the question:

Modal Logic

Modal logic is a logic to reason about necessity and possibility of future events that goes back to Aristotle. As an example of modal logic reasoning, Aristotle poses the question:

Will there be a sea battle tomorrow?

Modal Logic

Modal logic is a logic to reason about necessity and possibility of future events that goes back to Aristotle. As an example of modal logic reasoning, Aristotle poses the question:

Will there be a sea battle tomorrow?

Such an event is a property P about some future state of the "world,"

Modal Logic

Modal logic is a logic to reason about necessity and possibility of future events that goes back to Aristotle. As an example of modal logic reasoning, Aristotle poses the question:

Will there be a sea battle tomorrow?

Such an event is a property P about some future state of the "world," and there are two so-called modalities about property P:

Modal Logic

Modal logic is a logic to reason about necessity and possibility of future events that goes back to Aristotle. As an example of modal logic reasoning, Aristotle poses the question:

Will there be a sea battle tomorrow?

Such an event is a property P about some future state of the "world," and there are two so-called modalities about property P:

1. $\Box P$, read, necessarily P, or always P, means that P will always be the case in the future.

Modal Logic

Modal logic is a logic to reason about necessity and possibility of future events that goes back to Aristotle. As an example of modal logic reasoning, Aristotle poses the question:

Will there be a sea battle tomorrow?

Such an event is a property P about some future state of the "world," and there are two so-called modalities about property P:

1. $\Box P$, read, necessarily P, or always P, means that P will always be the case in the future. For example, $\Box\ 2+2=4$.

Modal Logic

Modal logic is a logic to reason about necessity and possibility of future events that goes back to Aristotle. As an example of modal logic reasoning, Aristotle poses the question:

Will there be a sea battle tomorrow?

Such an event is a property P about some future state of the "world," and there are two so-called modalities about property P:

1. $\Box P$, read, necessarily P, or always P, means that P will always be the case in the future. For example, $\Box\, 2 + 2 = 4$.
2. $\Diamond P$, read, possibly P, means that there is some possible future state of the world in which P holds.

Modal Logic

Modal logic is a logic to reason about necessity and possibility of future events that goes back to Aristotle. As an example of modal logic reasoning, Aristotle poses the question:

Will there be a sea battle tomorrow?

Such an event is a property P about some future state of the "world," and there are two so-called modalities about property P:

1. $\Box P$, read, necessarily P, or always P, means that P will always be the case in the future. For example, $\Box\, 2 + 2 = 4$.
2. $\Diamond P$, read, possibly P, means that there is some possible future state of the world in which P holds. For example, a possible state of the world tomorrow in which there will be a sea battle.

Modal Logic

Modal logic is a logic to reason about necessity and possibility of future events that goes back to Aristotle. As an example of modal logic reasoning, Aristotle poses the question:

Will there be a sea battle tomorrow?

Such an event is a property P about some future state of the "world," and there are two so-called modalities about property P:

1. $\Box P$, read, necessarily P, or always P, means that P will always be the case in the future. For example, $\Box\, 2 + 2 = 4$.
2. $\Diamond P$, read, possibly P, means that there is some possible future state of the world in which P holds. For example, a possible state of the world tomorrow in which there will be a sea battle.

Following Saul Kripke, analytic philosophers model this with a so-called possible worlds semantics.

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \to_Q)$,

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \to_{\mathcal{Q}})$, where Q is a set of states and $\to_{\mathcal{Q}} \subseteq Q \times Q$ is a state transition relation.

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \to_{\mathcal{Q}})$, where Q is a set of states and $\to_{\mathcal{Q}} \subseteq Q \times Q$ is a state transition relation. Then, given a set Π of property names, the meaning of each name $p \in \Pi$ in $(Q, \to_{\mathcal{Q}})$ is given by a property meaning function $\_{\mathcal{Q}} : \Pi \ni p \mapsto p_{\mathcal{Q}} \in \mathcal{P}(Q)$.

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \rightarrow_{\mathcal{Q}})$, where Q is a set of states and $\rightarrow_{\mathcal{Q}} \subseteq Q \times Q$ is a state transition relation. Then, given a set Π of property names, the meaning of each name $p \in \Pi$ in $(Q, \rightarrow_{\mathcal{Q}})$ is given by a property meaning function $\_{\mathcal{Q}} : \Pi \ni p \mapsto p_{\mathcal{Q}} \in \mathcal{P}(Q)$.

A Kripke structure is just a triple $\mathcal{Q} = (Q, \rightarrow_{\mathcal{Q}}, \_{\mathcal{Q}})$,

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \to_{\mathcal{Q}})$, where Q is a set of states and $\to_{\mathcal{Q}} \subseteq Q \times Q$ is a state transition relation. Then, given a set Π of property names, the meaning of each name $p \in \Pi$ in $(Q, \to_{\mathcal{Q}})$ is given by a property meaning function $\_{\mathcal{Q}} : \Pi \ni p \mapsto p_{\mathcal{Q}} \in \mathcal{P}(Q)$.

A Kripke structure is just a triple $\mathcal{Q} = (Q, \to_{\mathcal{Q}}, \_{\mathcal{Q}})$, with $\_{\mathcal{Q}}$ interpreting the names Π in the transition system $(Q, \to_{\mathcal{Q}})$.

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \to_{\mathcal{Q}})$, where Q is a set of states and $\to_{\mathcal{Q}} \subseteq Q \times Q$ is a state transition relation. Then, given a set Π of property names, the meaning of each name $p \in \Pi$ in $(Q, \to_{\mathcal{Q}})$ is given by a property meaning function $\_{\mathcal{Q}} : \Pi \ni p \mapsto p_{\mathcal{Q}} \in \mathcal{P}(Q)$.

A Kripke structure is just a triple $\mathcal{Q} = (Q, \to_{\mathcal{Q}}, \_{\mathcal{Q}})$, with $\_{\mathcal{Q}}$ interpreting the names Π in the transition system $(Q, \to_{\mathcal{Q}})$.

The S4-meaning in $\mathcal{Q}$ of a modal logic formula φ is relative to a chosen set of initial states $I \subseteq Q$.

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \to_{\mathcal{Q}})$, where Q is a set of states and $\to_{\mathcal{Q}} \subseteq Q \times Q$ is a state transition relation. Then, given a set Π of property names, the meaning of each name $p \in \Pi$ in $(Q, \to_{\mathcal{Q}})$ is given by a property meaning function $\_{\mathcal{Q}} : \Pi \ni p \mapsto p_{\mathcal{Q}} \in \mathcal{P}(Q)$.

A Kripke structure is just a triple $\mathcal{Q} = (Q, \to_{\mathcal{Q}}, \_{\mathcal{Q}})$, with $\_{\mathcal{Q}}$ interpreting the names Π in the transition system $(Q, \to_{\mathcal{Q}})$.

The S4-meaning in $\mathcal{Q}$ of a modal logic formula φ is relative to a chosen set of initial states $I \subseteq Q$. It is defined by a semantic relation of the form: $\mathcal{Q}, I \models_{S4} \varphi$ as follows

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \to_{\mathcal{Q}})$, where Q is a set of states and $\to_{\mathcal{Q}} \subseteq Q \times Q$ is a state transition relation. Then, given a set Π of property names, the meaning of each name $p \in \Pi$ in $(Q, \to_{\mathcal{Q}})$ is given by a property meaning function $\_{\mathcal{Q}} : \Pi \ni p \mapsto p_{\mathcal{Q}} \in \mathcal{P}(Q)$.

A Kripke structure is just a triple $\mathcal{Q} = (Q, \to_{\mathcal{Q}}, \_{\mathcal{Q}})$, with $\_{\mathcal{Q}}$ interpreting the names Π in the transition system $(Q, \to_{\mathcal{Q}})$.

The S4-meaning in $\mathcal{Q}$ of a modal logic formula φ is relative to a chosen set of initial states $I \subseteq Q$. It is defined by a semantic relation of the form: $\mathcal{Q}, I \models_{S4} \varphi$ as follows (I will only focus on formulas $\varphi = \Box B$ or $\varphi = \Diamond B$,

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \to_Q)$, where Q is a set of states and $\to_Q \subseteq Q \times Q$ is a state transition relation. Then, given a set Π of property names, the meaning of each name $p \in \Pi$ in $(Q, \to_Q)$ is given by a property meaning function $\_{}_{\mathcal{Q}} : \Pi \ni p \mapsto p_{\mathcal{Q}} \in \mathcal{P}(Q)$.

A Kripke structure is just a triple $\mathcal{Q} = (Q, \to_{\mathcal{Q}}, \_{}_{\mathcal{Q}})$, with $\_{}_{\mathcal{Q}}$ interpreting the names Π in the transition system $(Q, \to_{\mathcal{Q}})$.

The S4-meaning in $\mathcal{Q}$ of a modal logic formula φ is relative to a chosen set of initial states $I \subseteq Q$. It is defined by a semantic relation of the form: $\mathcal{Q}, I \models_{S4} \varphi$ as follows (I will only focus on formulas $\varphi = \Box B$ or $\varphi = \Diamond B$, with B a Boolean combination of names $p_i \in \Pi$,

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \to_{\mathcal{Q}})$, where Q is a set of states and $\to_{\mathcal{Q}} \subseteq Q \times Q$ is a state transition relation. Then, given a set Π of property names, the meaning of each name $p \in \Pi$ in $(Q, \to_{\mathcal{Q}})$ is given by a property meaning function $\_{\mathcal{Q}} : \Pi \ni p \mapsto p_{\mathcal{Q}} \in \mathcal{P}(Q)$.

A Kripke structure is just a triple $\mathcal{Q} = (Q, \to_{\mathcal{Q}}, \_{\mathcal{Q}})$, with $\_{\mathcal{Q}}$ interpreting the names Π in the transition system $(Q, \to_{\mathcal{Q}})$.

The S4-meaning in $\mathcal{Q}$ of a modal logic formula φ is relative to a chosen set of initial states $I \subseteq Q$. It is defined by a semantic relation of the form: $\mathcal{Q}, I \models_{S4} \varphi$ as follows (I will only focus on formulas $\varphi = \Box B$ or $\varphi = \Diamond B$, with B a Boolean combination of names $p_i \in \Pi$, whose meaning is defined by:

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \to_{\mathcal{Q}})$, where Q is a set of states and $\to_{\mathcal{Q}} \subseteq Q \times Q$ is a state transition relation. Then, given a set Π of property names, the meaning of each name $p \in \Pi$ in $(Q, \to_{\mathcal{Q}})$ is given by a property meaning function $\_{\mathcal{Q}} : \Pi \ni p \mapsto p_{\mathcal{Q}} \in \mathcal{P}(Q)$.

A Kripke structure is just a triple $\mathcal{Q} = (Q, \to_{\mathcal{Q}}, \_{\mathcal{Q}})$, with $\_{\mathcal{Q}}$ interpreting the names Π in the transition system $(Q, \to_{\mathcal{Q}})$.

The S4-meaning in $\mathcal{Q}$ of a modal logic formula φ is relative to a chosen set of initial states $I \subseteq Q$. It is defined by a semantic relation of the form: $\mathcal{Q}, I \models_{S4} \varphi$ as follows (I will only focus on formulas $\varphi = \Box B$ or $\varphi = \Diamond B$, with B a Boolean combination of names $p_i \in \Pi$, whose meaning is defined by: $(\neg B)_{\mathcal{Q}} =_{def} Q \setminus B_{\mathcal{Q}}$,

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \rightarrow_{\mathcal{Q}})$, where Q is a set of states and $\rightarrow_{\mathcal{Q}} \subseteq Q \times Q$ is a state transition relation. Then, given a set Π of property names, the meaning of each name $p \in \Pi$ in $(Q, \rightarrow_{\mathcal{Q}})$ is given by a property meaning function $\_{}_{\mathcal{Q}} : \Pi \ni p \mapsto p_{\mathcal{Q}} \in \mathcal{P}(Q)$.

A Kripke structure is just a triple $\mathcal{Q} = (Q, \rightarrow_{\mathcal{Q}}, \_{}_{\mathcal{Q}})$, with $\_{}_{\mathcal{Q}}$ interpreting the names Π in the transition system $(Q, \rightarrow_{\mathcal{Q}})$.

The S4-meaning in $\mathcal{Q}$ of a modal logic formula φ is relative to a chosen set of initial states $I \subseteq Q$. It is defined by a semantic relation of the form: $\mathcal{Q}, I \models_{S4} \varphi$ as follows (I will only focus on formulas $\varphi = \Box B$ or $\varphi = \Diamond B$, with B a Boolean combination of names $p_i \in \Pi$, whose meaning is defined by: $(\neg B)_{\mathcal{Q}} =_{def} Q \setminus B_{\mathcal{Q}}$, $(A \vee B)_{\mathcal{Q}} =_{def} A_{\mathcal{Q}} \cup B_{\mathcal{Q}}$, and

Kripke Semantics of Modal Logic

The late Saul Kripke proposed a simple mathematical semantics for modal logic in which the "states of the world" are the states of a transition system $(Q, \to_{\mathcal{Q}})$, where Q is a set of states and $\to_{\mathcal{Q}} \subseteq Q \times Q$ is a state transition relation. Then, given a set Π of property names, the meaning of each name $p \in \Pi$ in $(Q, \to_{\mathcal{Q}})$ is given by a property meaning function $\_{\mathcal{Q}} : \Pi \ni p \mapsto p_{\mathcal{Q}} \in \mathcal{P}(Q)$.

A Kripke structure is just a triple $\mathcal{Q} = (Q, \to_{\mathcal{Q}}, \_{\mathcal{Q}})$, with $\_{\mathcal{Q}}$ interpreting the names Π in the transition system $(Q, \to_{\mathcal{Q}})$.

The S4-meaning in $\mathcal{Q}$ of a modal logic formula φ is relative to a chosen set of initial states $I \subseteq Q$. It is defined by a semantic relation of the form: $\mathcal{Q}, I \models_{S4} \varphi$ as follows (I will only focus on formulas $\varphi = \Box B$ or $\varphi = \Diamond B$, with B a Boolean combination of names $p_i \in \Pi$, whose meaning is defined by: $(\neg B)_{\mathcal{Q}} =_{def} Q \setminus B_{\mathcal{Q}}$, $(A \vee B)_{\mathcal{Q}} =_{def} A_{\mathcal{Q}} \cup B_{\mathcal{Q}}$, and $(A \wedge B)_{\mathcal{Q}} =_{def} A_{\mathcal{Q}} \cap B_{\mathcal{Q}}$):

Kripke Semantics of Modal Logic (II)

$$\mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow_{def} \quad \forall q_0 \in I,\ \forall q \in Q,\ q_0 \rightarrow^*_{\mathcal{Q}} q \ \Rightarrow\ q \in B_{\mathcal{Q}}$$

Kripke Semantics of Modal Logic (II)

$$\mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow_{def} \quad \forall q_0 \in I,\ \forall q \in Q,\ q_0 \rightarrow^*_{\mathcal{Q}} q \ \Rightarrow \ q \in B_{\mathcal{Q}}$$

$$\mathcal{Q}, I \models_{S4} \Diamond B \quad \Leftrightarrow_{def} \quad \exists q_0 \in I,\ \exists q \in Q,\ q_0 \rightarrow^*_{\mathcal{Q}} q \ \wedge q \in B_{\mathcal{Q}}$$

Kripke Semantics of Modal Logic (II)

$$\mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow_{def} \quad \forall q_0 \in I,\ \forall q \in Q,\ q_0 \rightarrow^*_{\mathcal{Q}} q \ \Rightarrow \ q \in B_{\mathcal{Q}}$$

$$\mathcal{Q}, I \models_{S4} \Diamond B \quad \Leftrightarrow_{def} \quad \exists q_0 \in I,\ \exists q \in Q,\ q_0 \rightarrow^*_{\mathcal{Q}} q \ \wedge q \in B_{\mathcal{Q}}$$

Note the striking duality between $\Box$ and $\Diamond$,

Kripke Semantics of Modal Logic (II)

$$\mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow_{def} \quad \forall q_0 \in I,\ \forall q \in Q,\ q_0 \rightarrow^*_{\mathcal{Q}} q \ \Rightarrow\ q \in B_{\mathcal{Q}}$$

$$\mathcal{Q}, I \models_{S4} \Diamond B \quad \Leftrightarrow_{def} \quad \exists q_0 \in I,\ \exists q \in Q,\ q_0 \rightarrow^*_{\mathcal{Q}} q \ \wedge q \in B_{\mathcal{Q}}$$

Note the striking duality between $\Box$ and $\Diamond$, namely,

$$(\dagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond \neg\, B$$

Kripke Semantics of Modal Logic (II)

$$\mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow_{def} \quad \forall q_0 \in I,\ \forall q \in Q,\ q_0 \to^{*}_{\mathcal{Q}} q \ \Rightarrow \ q \in B_{\mathcal{Q}}$$

$$\mathcal{Q}, I \models_{S4} \Diamond B \quad \Leftrightarrow_{def} \quad \exists q_0 \in I,\ \exists q \in Q,\ q_0 \to^{*}_{\mathcal{Q}} q \ \wedge q \in B_{\mathcal{Q}}$$

Note the striking duality between $\Box$ and $\Diamond$, namely,

$$(\dagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond \neg B$$

That is, B is necessary iff $\neg B$ is impossible, and therefore,

Kripke Semantics of Modal Logic (II)

$$\mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow_{def} \quad \forall q_0 \in I,\ \forall q \in Q,\ q_0 \to^*_{\mathcal{Q}} q \ \Rightarrow \ q \in B_{\mathcal{Q}}$$

$$\mathcal{Q}, I \models_{S4} \Diamond B \quad \Leftrightarrow_{def} \quad \exists q_0 \in I,\ \exists q \in Q,\ q_0 \to^*_{\mathcal{Q}} q \ \wedge q \in B_{\mathcal{Q}}$$

Note the striking duality between $\Box$ and $\Diamond$, namely,

$$(\dagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond \neg B$$

That is, B is necessary iff $\neg B$ is impossible, and therefore,

$$(\ddagger) \quad \mathcal{Q}, I \models_{S4} \Box B$$

Kripke Semantics of Modal Logic (II)

$$\mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow_{def} \quad \forall q_0 \in I,\ \forall q \in Q,\ q_0 \to^*_{\mathcal{Q}} q \ \Rightarrow \ q \in B_{\mathcal{Q}}$$

$$\mathcal{Q}, I \models_{S4} \Diamond B \quad \Leftrightarrow_{def} \quad \exists q_0 \in I,\ \exists q \in Q,\ q_0 \to^*_{\mathcal{Q}} q \ \wedge q \in B_{\mathcal{Q}}$$

Note the striking duality between $\Box$ and $\Diamond$, namely,

$$(\dagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond\neg\ B$$

That is, B is necessary iff $\neg\ B$ is impossible, and therefore,

$$(\ddagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond\neg\ B$$

Kripke Semantics of Modal Logic (II)

$$\mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow_{def} \quad \forall q_0 \in I,\ \forall q \in Q,\ q_0 \rightarrow^*_{\mathcal{Q}} q \ \Rightarrow\ q \in B_{\mathcal{Q}}$$

$$\mathcal{Q}, I \models_{S4} \Diamond B \quad \Leftrightarrow_{def} \quad \exists q_0 \in I,\ \exists q \in Q,\ q_0 \rightarrow^*_{\mathcal{Q}} q \ \wedge q \in B_{\mathcal{Q}}$$

Note the striking duality between $\Box$ and $\Diamond$, namely,

$$(\dagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond\neg\, B$$

That is, B is necessary iff $\neg\, B$ is impossible, and therefore,

$$(\ddagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond\neg\, B \quad \Leftrightarrow \quad \mathcal{Q}, I \models_{S4} \neg\Diamond\neg\, B$$

Kripke Semantics of Modal Logic (II)

$$\mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow_{def} \quad \forall q_0 \in I,\ \forall q \in Q,\ q_0 \to^*_{\mathcal{Q}} q \ \Rightarrow \ q \in B_{\mathcal{Q}}$$

$$\mathcal{Q}, I \models_{S4} \Diamond B \quad \Leftrightarrow_{def} \quad \exists q_0 \in I,\ \exists q \in Q,\ q_0 \to^*_{\mathcal{Q}} q \ \wedge q \in B_{\mathcal{Q}}$$

Note the striking duality between $\Box$ and $\Diamond$, namely,

$$(\dagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond \neg B$$

That is, B is necessary iff $\neg B$ is impossible, and therefore,

$$(\ddagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond \neg B \quad \Leftrightarrow \quad \mathcal{Q}, I \models_{S4} \neg \Diamond \neg B$$

That is, we have duality equivalences:

Kripke Semantics of Modal Logic (II)

$$\mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow_{def} \quad \forall q_0 \in I,\ \forall q \in Q,\ q_0 \rightarrow^*_{\mathcal{Q}} q \ \Rightarrow \ q \in B_{\mathcal{Q}}$$

$$\mathcal{Q}, I \models_{S4} \Diamond B \quad \Leftrightarrow_{def} \quad \exists q_0 \in I,\ \exists q \in Q,\ q_0 \rightarrow^*_{\mathcal{Q}} q \ \wedge q \in B_{\mathcal{Q}}$$

Note the striking duality between $\Box$ and $\Diamond$, namely,

$$(\dagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond \neg\, B$$

That is, B is necessary iff $\neg\, B$ is impossible, and therefore,

$$(\ddagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond \neg\, B \quad \Leftrightarrow \quad \mathcal{Q}, I \models_{S4} \neg \Diamond \neg\, B$$

That is, we have duality equivalences: $\Box \equiv \neg \Diamond \neg$ and $\Diamond \equiv \neg \Box \neg$,

Kripke Semantics of Modal Logic (II)

$$\mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow_{def} \quad \forall q_0 \in I,\ \forall q \in Q,\ q_0 \to_{\mathcal{Q}}^{*} q \ \Rightarrow \ q \in B_{\mathcal{Q}}$$

$$\mathcal{Q}, I \models_{S4} \Diamond B \quad \Leftrightarrow_{def} \quad \exists q_0 \in I,\ \exists q \in Q,\ q_0 \to_{\mathcal{Q}}^{*} q \ \wedge q \in B_{\mathcal{Q}}$$

Note the striking duality between $\Box$ and $\Diamond$, namely,

$$(\dagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond\neg\, B$$

That is, B is necessary iff $\neg\, B$ is impossible, and therefore,

$$(\ddagger) \quad \mathcal{Q}, I \models_{S4} \Box B \quad \Leftrightarrow \quad \mathcal{Q}, I \not\models_{S4} \Diamond\neg\, B \quad \Leftrightarrow \quad \mathcal{Q}, I \models_{S4} \neg\Diamond\neg\, B$$

That is, we have duality equivalences: $\Box \equiv \neg\Diamond\neg$ and $\Diamond \equiv \neg\Box\neg$, like the duality equivalences defining $\forall$ in terms of $\exists$ or viceversa.

A Modal Logic for Rewrite Theories

Consider a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying requirements (1)–(2), and a distinguished sort, say St, of states.

A Modal Logic for Rewrite Theories

Consider a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying requirements (1)–(2), and a distinguished sort, say St, of states. Then, if φ is a modal logic formula, and I is a set of initial states $I \subseteq C_{\Sigma/E,B,St}$,

A Modal Logic for Rewrite Theories

Consider a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying requirements (1)–(2), and a distinguished sort, say St, of states. Then, if φ is a modal logic formula, and I is a set of initial states $I \subseteq C_{\Sigma/E,B,St}$, we define $\mathcal{R}, I \models \varphi$ by the following chain of defining equivalences:

A Modal Logic for Rewrite Theories

Consider a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying requirements (1)–(2), and a distinguished sort, say St, of states. Then, if φ is a modal logic formula, and I is a set of initial states $I \subseteq C_{\Sigma/E,B,St}$, we define $\mathcal{R}, I \models \varphi$ by the following chain of defining equivalences:

$$\mathcal{R}, I \models \varphi \quad \Leftrightarrow_{def}$$

A Modal Logic for Rewrite Theories

Consider a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying requirements (1)–(2), and a distinguished sort, say St, of states. Then, if φ is a modal logic formula, and I is a set of initial states $I \subseteq C_{\Sigma/E,B,St}$, we define $\mathcal{R}, I \models \varphi$ by the following chain of defining equivalences:

$$\mathcal{R}, I \models \varphi \quad \Leftrightarrow_{def} \quad \mathbb{C}_{\mathcal{R}}, I \models \varphi \quad \Leftrightarrow_{def}$$

A Modal Logic for Rewrite Theories

Consider a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying requirements (1)–(2), and a distinguished sort, say St, of states. Then, if φ is a modal logic formula, and I is a set of initial states $I \subseteq C_{\Sigma/E,B,St}$, we define $\mathcal{R}, I \models \varphi$ by the following chain of defining equivalences:

$$\mathcal{R}, I \models \varphi \quad \Leftrightarrow_{def} \quad \mathbb{C}_{\mathcal{R}}, I \models \varphi \quad \Leftrightarrow_{def} \quad (C_{\Sigma/\vec{E},B,St}, \rightarrow_{\mathbb{C}_{\mathcal{R}}}, \neg_{\mathbb{C}_{\mathcal{R}}}), I \models_{S4} \varphi.$$

A Modal Logic for Rewrite Theories

Consider a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying requirements (1)–(2), and a distinguished sort, say St, of states. Then, if φ is a modal logic formula, and I is a set of initial states $I \subseteq C_{\Sigma/E,B,St}$, we define $\mathcal{R}, I \models \varphi$ by the following chain of defining equivalences:

$$\mathcal{R}, I \models \varphi \quad \Leftrightarrow_{def} \quad \mathbb{C}_{\mathcal{R}}, I \models \varphi \quad \Leftrightarrow_{def} \quad (C_{\Sigma/\vec{E},B,St}, \to_{\mathbb{C}_{\mathcal{R}}}, \vdash_{\mathbb{C}_{\mathcal{R}}}), I \models_{S4} \varphi.$$

That is, the transition system on which we give a Kripke semantics to φ is $(\mathbb{C}_{\Sigma/\vec{E},B,St}, \to_{\mathbb{C}_{\mathcal{R}}})$,

A Modal Logic for Rewrite Theories

Consider a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying requirements (1)–(2), and a distinguished sort, say St, of states. Then, if φ is a modal logic formula, and I is a set of initial states $I \subseteq C_{\Sigma/E,B,St}$, we define $\mathcal{R}, I \models \varphi$ by the following chain of defining equivalences:

$$\mathcal{R}, I \models \varphi \quad \Leftrightarrow_{def} \quad \mathbb{C}_{\mathcal{R}}, I \models \varphi \quad \Leftrightarrow_{def} \quad (C_{\Sigma/\vec{E},B,St}, \rightarrow_{\mathbb{C}_{\mathcal{R}}}, \cdot\mathbb{C}_{\mathcal{R}}), I \models_{S4} \varphi.$$

That is, the transition system on which we give a Kripke semantics to φ is $(\mathbb{C}_{\Sigma/\vec{E},B,St}, \rightarrow_{\mathbb{C}_{\mathcal{R}}})$, where φ will mention some property names $p_i \in \Pi$,

A Modal Logic for Rewrite Theories

Consider a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying requirements (1)–(2), and a distinguished sort, say St, of states. Then, if φ is a modal logic formula, and I is a set of initial states $I \subseteq C_{\Sigma/E,B,St}$, we define $\mathcal{R}, I \models \varphi$ by the following chain of defining equivalences:

$$\mathcal{R}, I \models \varphi \quad \Leftrightarrow_{def} \quad \mathbb{C}_{\mathcal{R}}, I \models \varphi \quad \Leftrightarrow_{def} \quad (C_{\Sigma/\vec{E},B,St}, \rightarrow_{\mathbb{C}_{\mathcal{R}}}, \_{\mathbb{C}_{\mathcal{R}}}), I \models_{S4} \varphi.$$

That is, the transition system on which we give a Kripke semantics to φ is $(\mathbb{C}_{\Sigma/\vec{E},B,St}, \rightarrow_{\mathbb{C}_{\mathcal{R}}})$, where φ will mention some property names $p_i \in \Pi$, with the meaning function $\_{\mathbb{C}_{\mathcal{R}}}$ interpreting each p_i as a subset $p_{\mathbb{C}_{\mathcal{R}}} \subseteq C_{\Sigma/\vec{E},B,St}$.

A Modal Logic for Rewrite Theories

Consider a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying requirements (1)–(2), and a distinguished sort, say St, of states. Then, if φ is a modal logic formula, and I is a set of initial states $I \subseteq C_{\Sigma/E,B,St}$, we define $\mathcal{R}, I \models \varphi$ by the following chain of defining equivalences:

$$\mathcal{R}, I \models \varphi \quad \Leftrightarrow_{def} \quad \mathbb{C}_{\mathcal{R}}, I \models \varphi \quad \Leftrightarrow_{def} \quad (C_{\Sigma/\vec{E},B,St}, \rightarrow_{\mathbb{C}_{\mathcal{R}}}, \_{\mathbb{C}_{\mathcal{R}}}), I \models_{S4} \varphi.$$

That is, the transition system on which we give a Kripke semantics to φ is $(\mathbb{C}_{\Sigma/\vec{E},B,St}, \rightarrow_{\mathbb{C}_{\mathcal{R}}})$, where φ will mention some property names $p_i \in \Pi$, with the meaning function $\_{\mathbb{C}_{\mathcal{R}}}$ interpreting each p_i as a subset $p_{\mathbb{C}_{\mathcal{R}}} \subseteq C_{\Sigma/\vec{E},B,St}$.

In practice we shall want to interpret each $p \in \Pi$ as a computable subset $p_{\mathbb{C}_{\mathcal{R}}} \subseteq C_{\Sigma/\vec{E},B,St}$,

A Modal Logic for Rewrite Theories

Consider a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying requirements (1)–(2), and a distinguished sort, say St, of states. Then, if φ is a modal logic formula, and I is a set of initial states $I \subseteq C_{\Sigma/E,B,St}$, we define $\mathcal{R}, I \models \varphi$ by the following chain of defining equivalences:

$$\mathcal{R}, I \models \varphi \quad \Leftrightarrow_{def} \quad \mathbb{C}_{\mathcal{R}}, I \models \varphi \quad \Leftrightarrow_{def} \quad (C_{\Sigma/\vec{E},B,St}, \rightarrow_{\mathbb{C}_{\mathcal{R}}}, \_{\mathbb{C}_{\mathcal{R}}}), I \models_{S4} \varphi.$$

That is, the transition system on which we give a Kripke semantics to φ is $(\mathbb{C}_{\Sigma/\vec{E},B,St}, \rightarrow_{\mathbb{C}_{\mathcal{R}}})$, where φ will mention some property names $p_i \in \Pi$, with the meaning function $\_{\mathbb{C}_{\mathcal{R}}}$ interpreting each p_i as a subset $p_{\mathbb{C}_{\mathcal{R}}} \subseteq C_{\Sigma/\vec{E},B,St}$.

In practice we shall want to interpret each $p \in \Pi$ as a computable subset $p_{\mathbb{C}_{\mathcal{R}}} \subseteq C_{\Sigma/\vec{E},B,St}$, i.e., a subset $p_{\mathbb{C}_{\mathcal{R}}} \subseteq C_{\Sigma/\vec{E},B,St}$ such that, given any $[u] \in C_{\Sigma/\vec{E},B,St}$,

A Modal Logic for Rewrite Theories

Consider a rewrite theory $\mathcal{R} = (\Sigma, E \cup B, R)$ satisfying requirements (1)–(2), and a distinguished sort, say St, of states. Then, if φ is a modal logic formula, and I is a set of initial states $I \subseteq C_{\Sigma/E,B,St}$, we define $\mathcal{R}, I \models \varphi$ by the following chain of defining equivalences:

$$\mathcal{R}, I \models \varphi \quad \Leftrightarrow_{def} \quad \mathbb{C}_{\mathcal{R}}, I \models \varphi \quad \Leftrightarrow_{def} \quad (C_{\Sigma/\vec{E},B,St}, \to_{\mathbb{C}_{\mathcal{R}}}, \_{\mathbb{C}_{\mathcal{R}}}), I \models_{S4} \varphi.$$

That is, the transition system on which we give a Kripke semantics to φ is $(\mathbb{C}_{\Sigma/\vec{E},B,St}, \to_{\mathbb{C}_{\mathcal{R}}})$, where φ will mention some property names $p_i \in \Pi$, with the meaning function $\_{\mathbb{C}_{\mathcal{R}}}$ interpreting each p_i as a subset $p_{\mathbb{C}_{\mathcal{R}}} \subseteq C_{\Sigma/\vec{E},B,St}$.

In practice we shall want to interpret each $p \in \Pi$ as a computable subset $p_{\mathbb{C}_{\mathcal{R}}} \subseteq C_{\Sigma/\vec{E},B,St}$, i.e., a subset $p_{\mathbb{C}_{\mathcal{R}}} \subseteq C_{\Sigma/\vec{E},B,St}$ such that, given any $[u] \in C_{\Sigma/\vec{E},B,St}$, we can effectively decide whether $[u] \in p_{\mathbb{C}_{\mathcal{R}}}$ or $[u] \notin p_{\mathbb{C}_{\mathcal{R}}}$.

Invariants

Invariants are the most basic safety properties that a concurrent system specified by a rewrite theory $\mathcal{R}$ can satisfy.

Invariants

Invariants are the most basic safety properties that a concurrent system specified by a rewrite theory $\mathcal{R}$ can satisfy. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called an invariant of $\mathcal{R}$ from initial states I iff

Invariants

Invariants are the most basic safety properties that a concurrent system specified by a rewrite theory $\mathcal{R}$ can satisfy. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called an invariant of $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Box p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

Invariants

Invariants are the most basic safety properties that a concurrent system specified by a rewrite theory $\mathcal{R}$ can satisfy. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called an invariant of $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Box p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

An invariant Q describes a "good" or "safe" state property that should always hold.

Invariants

Invariants are the most basic safety properties that a concurrent system specified by a rewrite theory $\mathcal{R}$ can satisfy. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called an invariant of $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Box p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

An invariant Q describes a "good" or "safe" state property that should always hold. Instead, its complement $\overline{Q}$ describes a set of "bad" or "unsafe" states that the system should never be in.

Invariants

Invariants are the most basic safety properties that a concurrent system specified by a rewrite theory $\mathcal{R}$ can satisfy. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called an invariant of $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Box p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

An invariant Q describes a "good" or "safe" state property that should always hold. Instead, its complement $\overline{Q}$ describes a set of "bad" or "unsafe" states that the system should never be in.

Q is a safety envelope:

Invariants

Invariants are the most basic safety properties that a concurrent system specified by a rewrite theory $\mathcal{R}$ can satisfy. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called an invariant of $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Box p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

An invariant Q describes a "good" or "safe" state property that should always hold. Instead, its complement $\overline{Q}$ describes a set of "bad" or "unsafe" states that the system should never be in.

Q is a safety envelope: Q is an invariant from I iff any state reachable for the set I of initial states is within the safety envelope Q.

Invariants

Invariants are the most basic safety properties that a concurrent system specified by a rewrite theory $\mathcal{R}$ can satisfy. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called an invariant of $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Box p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

An invariant Q describes a "good" or "safe" state property that should always hold. Instead, its complement $\overline{Q}$ describes a set of "bad" or "unsafe" states that the system should never be in.

Q is a safety envelope: Q is an invariant from I iff any state reachable for the set I of initial states is within the safety envelope Q. Thanks to the equivalence (†), $\mathcal{R}, I \models \Box p^Q \Leftrightarrow \mathcal{R}, I \not\models \Diamond\neg p^Q$,

Invariants

Invariants are the most basic safety properties that a concurrent system specified by a rewrite theory $\mathcal{R}$ can satisfy. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called an invariant of $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Box p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

An invariant Q describes a "good" or "safe" state property that should always hold. Instead, its complement $\overline{Q}$ describes a set of "bad" or "unsafe" states that the system should never be in.

Q is a safety envelope: Q is an invariant from I iff any state reachable for the set I of initial states is within the safety envelope Q. Thanks to the equivalence (†), $\mathcal{R}, I \models \Box p^Q \Leftrightarrow \mathcal{R}, I \not\models \Diamond \neg p^Q$, for I a single initial state *init*, this suggest using Maude's `search` command to search for states satisfying $\neg p^Q$.

Invariants

Invariants are the most basic safety properties that a concurrent system specified by a rewrite theory $\mathcal{R}$ can satisfy. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called an invariant of $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Box p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

An invariant Q describes a "good" or "safe" state property that should always hold. Instead, its complement $\overline{Q}$ describes a set of "bad" or "unsafe" states that the system should never be in.

Q is a safety envelope: Q is an invariant from I iff any state reachable for the set I of initial states is within the safety envelope Q. Thanks to the equivalence $(\dagger)$, $\mathcal{R}, I \models \Box p^Q \Leftrightarrow \mathcal{R}, I \not\models \Diamond \neg p^Q$, for I a single initial state *init*, this suggest using Maude's `search` command to search for states satisfying $\neg p^Q$. If no such states exist we will have verified Q.

Invariants

Invariants are the most basic safety properties that a concurrent system specified by a rewrite theory $\mathcal{R}$ can satisfy. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called an invariant of $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Box p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

An invariant Q describes a "good" or "safe" state property that should always hold. Instead, its complement $\overline{Q}$ describes a set of "bad" or "unsafe" states that the system should never be in.

Q is a safety envelope: Q is an invariant from I iff any state reachable for the set I of initial states is within the safety envelope Q. Thanks to the equivalence (†), $\mathcal{R}, I \models \Box p^Q \Leftrightarrow \mathcal{R}, I \not\models \Diamond \neg p^Q$, for I a single initial state *init*, this suggest using Maude's `search` command to search for states satisfying $\neg p^Q$. If no such states exist we will have verified Q. But how can we specify $\neg p^Q$?

Reachable Sets of States

We are also interested in verifying that a certain set of states Q is reachable in a concurrent system specified by a rewrite theory $\mathcal{R}$.

Reachable Sets of States

We are also interested in verifying that a certain set of states Q is reachable in a concurrent system specified by a rewrite theory $\mathcal{R}$. That is, we would like to check that some $q \in Q$ can be reached from some initial state along some computation path.

Reachable Sets of States

We are also interested in verifying that a certain set of states Q is reachable in a concurrent system specified by a rewrite theory $\mathcal{R}$. That is, we would like to check that some $q \in Q$ can be reached from some initial state along some computation path. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called a reachable in $\mathcal{R}$ from initial states I iff

Reachable Sets of States

We are also interested in verifying that a certain set of states Q is reachable in a concurrent system specified by a rewrite theory $\mathcal{R}$. That is, we would like to check that some $q \in Q$ can be reached from some initial state along some computation path. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called a reachable in $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Diamond p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

Reachable Sets of States

We are also interested in verifying that a certain set of states Q is reachable in a concurrent system specified by a rewrite theory $\mathcal{R}$. That is, we would like to check that some $q \in Q$ can be reached from some initial state along some computation path. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called a reachable in $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Diamond p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_{\mathcal{R}}} = Q$.

Note the duality between invariants and reachable states:

Reachable Sets of States

We are also interested in verifying that a certain set of states Q is reachable in a concurrent system specified by a rewrite theory $\mathcal{R}$. That is, we would like to check that some $q \in Q$ can be reached from some initial state along some computation path. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called a reachable in $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Diamond p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

Note the duality between invariants and reachable states: thanks to the equivalence (†) in page 9,

Reachable Sets of States

We are also interested in verifying that a certain set of states Q is reachable in a concurrent system specified by a rewrite theory $\mathcal{R}$. That is, we would like to check that some $q \in Q$ can be reached from some initial state along some computation path. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called a reachable in $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Diamond p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

Note the duality between invariants and reachable states: thanks to the equivalence (†) in page 9, Q is a invariant from initial states I iff its complement $Q \setminus C_{\Sigma/\vec{E},B,St}$ is unreachable from I.

Reachable Sets of States

We are also interested in verifying that a certain set of states Q is reachable in a concurrent system specified by a rewrite theory $\mathcal{R}$. That is, we would like to check that some $q \in Q$ can be reached from some initial state along some computation path. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called a reachable in $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Diamond p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_\mathcal{R}} = Q$.

Note the duality between invariants and reachable states: thanks to the equivalence (†) in page 9, Q is a invariant from initial states I iff its complement $Q \setminus C_{\Sigma/\vec{E},B,St}$ is unreachable from I.

for I a single initial state *init* we can use Maude's `search` command to `verify` that Q is reachable from *init* by searching for states in Q, where Q is specified by some state predicate p^Q.

Reachable Sets of States

We are also interested in verifying that a certain set of states Q is reachable in a concurrent system specified by a rewrite theory $\mathcal{R}$. That is, we would like to check that some $q \in Q$ can be reached from some initial state along some computation path. Given a chosen sort St of states and a set $I \subseteq C_{\Sigma/\vec{E},B,St}$ of initial states, $Q \subseteq C_{\Sigma/\vec{E},B,St}$ is called a reachable in $\mathcal{R}$ from initial states I iff $\mathcal{R}, I \models \Diamond p^Q$ for some property name $p^Q \in \Pi$ s.t. $p^Q_{\mathbb{C}_{\mathcal{R}}} = Q$.

Note the duality between invariants and reachable states: thanks to the equivalence (†) in page 9, Q is a invariant from initial states I iff its complement $Q \setminus C_{\Sigma/\vec{E},B,St}$ is unreachable from I.

for I a single initial state *init* we can use Maude's `search` command to `verify` that Q is reachable from *init* by searching for states in Q, where Q is specified by some state predicate p^Q. But how can we specify p^Q?

An Expressive Language Π for Modal Properties of $\mathbb{C}_{\mathcal{R}}$

What should we choose as our property names $p \in \Pi$ for $\mathbb{C}_{\mathcal{R}}$?

[1] φ could be allowed to be a $\Sigma \cup \Sigma_{aux}$-formula with Σ_{aux} auxiliary functions.

An Expressive Language Π for Modal Properties of $\mathbb{C}_{\mathcal{R}}$

What should we choose as our property names $p \in \Pi$ for $\mathbb{C}_{\mathcal{R}}$? An expressive property language already available to Maude users allows defining properties with constrained constructor patterns

[1] φ could be allowed to be a $\Sigma \cup \Sigma_{aux}$-formula with Σ_{aux} auxiliary functions.

An Expressive Language Π for Modal Properties of $\mathbb{C}_{\mathcal{R}}$

What should we choose as our property names $p \in \Pi$ for $\mathbb{C}_{\mathcal{R}}$? An expressive property language already available to Maude users allows defining properties with constrained constructor patterns of the form $u|\varphi$,

[1] φ could be allowed to be a $\Sigma \cup \Sigma_{aux}$-formula with Σ_{aux} auxiliary functions.

An Expressive Language Π for Modal Properties of $\mathbb{C}_{\mathcal{R}}$

What should we choose as our property names $p \in \Pi$ for $\mathbb{C}_{\mathcal{R}}$? An expressive property language already available to Maude users allows defining properties with constrained constructor patterns of the form $u|\varphi$, with u a constructor Ω-term of sort St with $vars(u) = \vec{x}$,

[1] φ could be allowed to be a $\Sigma \cup \Sigma_{aux}$-formula with Σ_{aux} auxiliary functions.

An Expressive Language Π for Modal Properties of $\mathbb{C}_{\mathcal{R}}$

What should we choose as our property names $p \in \Pi$ for $\mathbb{C}_{\mathcal{R}}$? An expressive property language already available to Maude users allows defining properties with constrained constructor patterns of the form $u|\varphi$, with u a constructor Ω-term of sort St with $vars(u) = \vec{x}$, and φ a conjunction of Σ-equalities[1] on variables $\vec{x}$.

---

[1] φ could be allowed to be a $\Sigma \cup \Sigma_{aux}$-formula with Σ_{aux} auxiliary functions.

An Expressive Language Π for Modal Properties of $\mathbb{C}_{\mathcal{R}}$

What should we choose as our property names $p \in \Pi$ for $\mathbb{C}_{\mathcal{R}}$? An expressive property language already available to Maude users allows defining properties with constrained constructor patterns of the form $u|\varphi$, with u a constructor Ω-term of sort St with $vars(u) = \vec{x}$, and φ a conjunction of Σ-equalities[1] on variables $\vec{x}$. The meaning function $\_{\mathbb{C}_{\mathcal{R}}}$ has the form: $\_{\mathbb{C}_{\mathcal{R}}} : (u|\varphi) \mapsto [\![u \mid \varphi]\!]$,

[1] φ could be allowed to be a $\Sigma \cup \Sigma_{aux}$-formula with Σ_{aux} auxiliary functions.

An Expressive Language Π for Modal Properties of $\mathbb{C}_{\mathcal{R}}$

What should we choose as our property names $p \in \Pi$ for $\mathbb{C}_{\mathcal{R}}$? An expressive property language already available to Maude users allows defining properties with constrained constructor patterns of the form $u|\varphi$, with u a constructor Ω-term of sort St with $vars(u) = \vec{x}$, and φ a conjunction of Σ-equalities[1] on variables $\vec{x}$. The meaning function $\_{\mathbb{C}_{\mathcal{R}}}$ has the form: $\_{\mathbb{C}_{\mathcal{R}}} : (u|\varphi) \mapsto [\![u \mid \varphi]\!]$, where, by definition, $[\![u \mid \varphi]\!]$ is the computable subset:

[1] φ could be allowed to be a $\Sigma \cup \Sigma_{aux}$-formula with Σ_{aux} auxiliary functions.

An Expressive Language Π for Modal Properties of $\mathbb{C}_{\mathcal{R}}$

What should we choose as our property names $p \in \Pi$ for $\mathbb{C}_{\mathcal{R}}$? An expressive property language already available to Maude users allows defining properties with constrained constructor patterns of the form $u|\varphi$, with u a constructor Ω-term of sort St with $vars(u) = \vec{x}$, and φ a conjunction of Σ-equalities[1] on variables $\vec{x}$. The meaning function $\__{\mathbb{C}_{\mathcal{R}}}$ has the form: $\__{\mathbb{C}_{\mathcal{R}}} : (u|\varphi) \mapsto [\![u \mid \varphi]\!]$, where, by definition, $[\![u \mid \varphi]\!]$ is the computable subset:

$$[\![u \mid \varphi]\!] = \{[v] \in C_{\Sigma/\vec{E},B,St} \mid \exists \rho \; s.t. \; v =_B u\rho \; \wedge \; E \cup B \vdash \varphi\rho\} \subseteq C_{\Sigma/\vec{E},B,St}$$

[1] φ could be allowed to be a $\Sigma \cup \Sigma_{aux}$-formula with Σ_{aux} auxiliary functions.

An Expressive Language Π for Modal Properties of $\mathbb{C}_{\mathcal{R}}$

What should we choose as our property names $p \in \Pi$ for $\mathbb{C}_{\mathcal{R}}$? An expressive property language already available to Maude users allows defining properties with constrained constructor patterns of the form $u|\varphi$, with u a constructor Ω-term of sort St with $vars(u) = \vec{x}$, and φ a conjunction of Σ-equalities[1] on variables $\vec{x}$. The meaning function $\_{\mathbb{C}_{\mathcal{R}}}$ has the form: $\_{\mathbb{C}_{\mathcal{R}}} : (u|\varphi) \mapsto [\![u \mid \varphi]\!]$, where, by definition, $[\![u \mid \varphi]\!]$ is the computable subset:

$$[\![u \mid \varphi]\!] = \{[v] \in C_{\Sigma/\vec{E},B,St} \mid \exists \rho \; s.t. \; v =_B u\rho \;\wedge\; E \cup B \vdash \varphi\rho\} \subseteq C_{\Sigma/\vec{E},B,St}$$

That is, $[\![u \mid \varphi]\!]$ is the set of ground instances of u that satisfy φ.

[1] φ could be allowed to be a $\Sigma \cup \Sigma_{aux}$-formula with Σ_{aux} auxiliary functions.

An Expressive Language Π for Modal Properties of $\mathbb{C}_{\mathcal{R}}$

What should we choose as our property names $p \in \Pi$ for $\mathbb{C}_{\mathcal{R}}$? An expressive property language already available to Maude users allows defining properties with constrained constructor patterns of the form $u|\varphi$, with u a constructor Ω-term of sort St with $vars(u) = \vec{x}$, and φ a conjunction of Σ-equalities[1] on variables $\vec{x}$. The meaning function $\_{}_{\mathbb{C}_{\mathcal{R}}}$ has the form: $\_{}_{\mathbb{C}_{\mathcal{R}}} : (u|\varphi) \mapsto [\![u \mid \varphi]\!]$, where, by definition, $[\![u \mid \varphi]\!]$ is the computable subset:

$$[\![u \mid \varphi]\!] = \{[v] \in C_{\Sigma/\vec{E},B,St} \mid \exists \rho \; s.t. \; v =_B u\rho \; \wedge \; E \cup B \vdash \varphi\rho\} \subseteq C_{\Sigma/\vec{E},B,St}$$

That is, $[\![u \mid \varphi]\!]$ is the set of ground instances of u that satisfy φ.

Property $u|\varphi$ is available to Maude users:

[1] φ could be allowed to be a $\Sigma \cup \Sigma_{aux}$-formula with Σ_{aux} auxiliary functions.

An Expressive Language Π for Modal Properties of $\mathbb{C}_{\mathcal{R}}$

What should we choose as our property names $p \in \Pi$ for $\mathbb{C}_{\mathcal{R}}$? An expressive property language already available to Maude users allows defining properties with constrained constructor patterns of the form $u|\varphi$, with u a constructor Ω-term of sort St with $vars(u) = \vec{x}$, and φ a conjunction of Σ-equalities[1] on variables $\vec{x}$. The meaning function $\__{\mathbb{C}_{\mathcal{R}}}$ has the form: $\__{\mathbb{C}_{\mathcal{R}}} : (u|\varphi) \mapsto [\![u \mid \varphi]\!]$, where, by definition, $[\![u \mid \varphi]\!]$ is the computable subset:

$$[\![u \mid \varphi]\!] = \{[v] \in C_{\Sigma/\vec{E},B,St} \mid \exists \rho \; s.t. \; v =_B u\rho \;\wedge\; E \cup B \vdash \varphi\rho\} \subseteq C_{\Sigma/\vec{E},B,St}$$

That is, $[\![u \mid \varphi]\!]$ is the set of ground instances of u that satisfy φ.

Property $u|\varphi$ is available to Maude users: in Maude's `search` command $u|\varphi$ is specified as the target pattern with syntax:

[1] φ could be allowed to be a $\Sigma \cup \Sigma_{aux}$-formula with Σ_{aux} auxiliary functions.

An Expressive Language Π for Modal Properties of $\mathbb{C}_{\mathcal{R}}$

What should we choose as our property names $p \in \Pi$ for $\mathbb{C}_{\mathcal{R}}$? An expressive property language already available to Maude users allows defining properties with constrained constructor patterns of the form $u|\varphi$, with u a constructor Ω-term of sort St with $vars(u) = \vec{x}$, and φ a conjunction of Σ-equalities[1] on variables $\vec{x}$. The meaning function $\_{}_{\mathbb{C}_{\mathcal{R}}}$ has the form: $\_{}_{\mathbb{C}_{\mathcal{R}}} : (u|\varphi) \mapsto [\![u \mid \varphi]\!]$, where, by definition, $[\![u \mid \varphi]\!]$ is the computable subset:

$$[\![u \mid \varphi]\!] = \{[v] \in C_{\Sigma/\vec{E},B,St} \mid \exists \rho \; s.t. \; v =_B u\rho \; \wedge \; E \cup B \vdash \varphi\rho\} \subseteq C_{\Sigma/\vec{E},B,St}$$

That is, $[\![u \mid \varphi]\!]$ is the set of ground instances of u that satisfy φ.

Property $u|\varphi$ is available to Maude users: in Maude's `search` command $u|\varphi$ is specified as the target pattern with syntax:

u `such that` φ `.`

[1] φ could be allowed to be a $\Sigma \cup \Sigma_{aux}$-formula with Σ_{aux} auxiliary functions.

Verifying Invariants and Reachability by Model Checking

We can apply all this to verify invariants $\mathcal{R}, init \models \Box p^Q$ (resp. reachable sets, $\mathcal{R}, init \models \Diamond p^Q$) by verifying through explicit-state model checking that $\mathcal{R}, init \not\models \Diamond\neg p^Q$ (resp. $\mathcal{R}, init \models \Diamond\neg p^Q$) using Maude's `search` command.

Verifying Invariants and Reachability by Model Checking

We can apply all this to verify invariants $\mathcal{R}, init \models \Box p^Q$ (resp. reachable sets, $\mathcal{R}, init \models \Diamond p^Q$) by verifying through explicit-state model checking that $\mathcal{R}, init \not\models \Diamond \neg p^Q$ (resp. $\mathcal{R}, init \models \Diamond \neg p^Q$) using Maude's `search` command. We just need to specify $\neg p^Q$ (resp. p^Q) as a disjunction of constrained constructor patterns:

Verifying Invariants and Reachability by Model Checking

We can apply all this to verify invariants $\mathcal{R}, init \models \Box p^Q$ (resp. reachable sets, $\mathcal{R}, init \models \Diamond p^Q$) by verifying through explicit-state model checking that $\mathcal{R}, init \not\models \Diamond \neg p^Q$ (resp. $\mathcal{R}, init \models \Diamond \neg p^Q$) using Maude's `search` command. We just need to specify $\neg p^Q$ (resp. p^Q) as a disjunction of constrained constructor patterns:

$$u_1 \mid \varphi_1 \;\vee\; \ldots \;\vee\; u_n \mid \varphi_n$$

Verifying Invariants and Reachability by Model Checking

We can apply all this to verify invariants $\mathcal{R}, init \models \Box p^Q$ (resp. reachable sets, $\mathcal{R}, init \models \Diamond p^Q$) by verifying through explicit-state model checking that $\mathcal{R}, init \not\models \Diamond \neg p^Q$ (resp. $\mathcal{R}, init \models \Diamond \neg p^Q$) using Maude's `search` command. We just need to specify $\neg p^Q$ (resp. p^Q) as a disjunction of constrained constructor patterns:

$$u_1 \mid \varphi_1 \ \vee \ \ldots \ \vee \ u_n \mid \varphi_n$$

$\mathcal{R}, init \models \Box p^Q$ (resp. $\mathcal{R}, init \models \Diamond p^Q$) will hold iff, for $1 \leq i \leq n$ (resp. some i, $1 \leq i \leq n$) the n Maude commands:

Verifying Invariants and Reachability by Model Checking

We can apply all this to verify invariants $\mathcal{R}, init \models \Box p^Q$ (resp. reachable sets, $\mathcal{R}, init \models \Diamond p^Q$) by verifying through explicit-state model checking that $\mathcal{R}, init \not\models \Diamond\neg p^Q$ (resp. $\mathcal{R}, init \models \Diamond\neg p^Q$) using Maude's `search` command. We just need to specify $\neg p^Q$ (resp. p^Q) as a disjunction of constrained constructor patterns:

$$u_1 \mid \varphi_1 \;\vee\; \ldots \;\vee\; u_n \mid \varphi_n$$

$\mathcal{R}, init \models \Box p^Q$ (resp. $\mathcal{R}, init \models \Diamond p^Q$) will hold iff, for $1 \leq i \leq n$ (resp. some i, $1 \leq i \leq n$) the n Maude commands:

`search init =>*` u_i `such that` φ_i `.`

Verifying Invariants and Reachability by Model Checking

We can apply all this to verify invariants $\mathcal{R}, init \models \Box p^Q$ (resp. reachable sets, $\mathcal{R}, init \models \Diamond p^Q$) by verifying through explicit-state model checking that $\mathcal{R}, init \not\models \Diamond\neg p^Q$ (resp. $\mathcal{R}, init \models \Diamond\neg p^Q$) using Maude's `search` command. We just need to specify $\neg p^Q$ (resp. p^Q) as a disjunction of constrained constructor patterns:

$$u_1 \mid \varphi_1 \;\vee\; \ldots \;\vee\; u_n \mid \varphi_n$$

$\mathcal{R}, init \models \Box p^Q$ (resp. $\mathcal{R}, init \models \Diamond p^Q$) will hold iff, for $1 \leq i \leq n$ (resp. some i, $1 \leq i \leq n$) the n Maude commands:

`search init =>*` u_i `such that` φ_i `.`

return the answer: `No solution .` (resp. one returns a solution).

Verifying Invariants and Reachability by Model Checking

We can apply all this to verify invariants $\mathcal{R}, init \models \Box p^Q$ (resp. reachable sets, $\mathcal{R}, init \models \Diamond p^Q$) by verifying through explicit-state model checking that $\mathcal{R}, init \not\models \Diamond\neg p^Q$ (resp. $\mathcal{R}, init \models \Diamond\neg p^Q$) using Maude's `search` command. We just need to specify $\neg p^Q$ (resp. p^Q) as a disjunction of constrained constructor patterns:

$$u_1 \mid \varphi_1 \;\vee\; \ldots \;\vee\; u_n \mid \varphi_n$$

$\mathcal{R}, init \models \Box p^Q$ (resp. $\mathcal{R}, init \models \Diamond p^Q$) will hold iff, for $1 \leq i \leq n$ (resp. some i, $1 \leq i \leq n$) the n Maude commands:

`search init =>*` u_i `such that` φ_i `.`

return the answer: `No solution .` (resp. one returns a solution).

Let us illustrate this explicit-state model checking method with `QLOCK`, a mutual exclusion protocol proposed by K. Futatsugi, where the number of processes is unbounded.

The QLOCK Mutual Exclusion Protocol

```
mod QLOCK is protecting NAT .
 sorts NatMSet NatList State .
 subsorts Nat < NatMSet NatList .
 op mt : -> NatMSet [ctor] .
 op _ _ : NatMSet NatMSet -> NatMSet [ctor assoc comm id: mt] .
 op nil : -> NatList [ctor] .
 op _;_ : NatList NatList -> NatList [ctor assoc id: nil] .
 op {_<_|_|_|_>} : NatMSet NatMSet NatMSet NatMSet NatList -> State [ctor] .
  op [_] : Nat -> NatMSet .  *** set of first n numbers
 op init : Nat -> State .   *** initial state, parametric on n

 vars n i j : Nat .  vars S U W C : NatMSet .  var Q : NatList .
 eq [0] = mt .
 eq [s(n)] = n [n] .
 eq init(n) = {[n] < mt | mt | mt | nil >} .

 rl [join] : {S i < U | W | C | Q >} => {S < U i | W | C | Q >} .
 rl [n2w] : {S < U i | W | C | Q >} => {S < U | W i | C | Q ; i >} .
 rl [w2c] : {S < U | W i | C | i ; Q >} => {S < U | W | C i | i ; Q >} .
 rl [c2n] : {S < U | W | C i | i ; Q >} => {S < U i | W | C | Q >} .
 rl [exit] : {S < U i | W | C | Q >} => {S i < U | W | C | Q >} .
endm
```

The QLOCK Mutual Exclusion Protocol (II)

Processes are numbers.

The QLOCK Mutual Exclusion Protocol (II)

Processes are numbers. There is a left area for processes outside the protocol, and a protocol area (inside angle brackets).

The QLOCK Mutual Exclusion Protocol (II)

Processes are numbers. There is a left area for processes outside the protocol, and a protocol area (inside angle brackets). Processes outside can join the protocol (`[join]`).

The QLOCK Mutual Exclusion Protocol (II)

Processes are numbers. There is a left area for processes outside the protocol, and a protocol area (inside angle brackets). Processes outside can join the protocol (`[join]`). The protocol area has normal, waiting, and critical stages, plus a waiting queue, where a process can register its name to signal that it wants to enter the critical section (`[n2w]`).

The QLOCK Mutual Exclusion Protocol (II)

Processes are numbers. There is a left area for processes outside the protocol, and a protocol area (inside angle brackets). Processes outside can join the protocol ([join]). The protocol area has normal, waiting, and critical stages, plus a waiting queue, where a process can register its name to signal that it wants to enter the critical section ([n2w]). When its name appears at the front of the queue, it is allowed to enter the critical section (rule [w2c]).

The QLOCK Mutual Exclusion Protocol (II)

Processes are numbers. There is a left area for processes outside the protocol, and a protocol area (inside angle brackets). Processes outside can join the protocol (`[join]`). The protocol area has normal, waiting, and critical stages, plus a waiting queue, where a process can register its name to signal that it wants to enter the critical section (`[n2w]`). When its name appears at the front of the queue, it is allowed to enter the critical section (rule `[w2c]`). When it has finished, it can go back to normal (rule `[c2n]`).

The QLOCK Mutual Exclusion Protocol (II)

Processes are numbers. There is a left area for processes outside the protocol, and a protocol area (inside angle brackets). Processes outside can join the protocol (`[join]`). The protocol area has normal, waiting, and critical stages, plus a waiting queue, where a process can register its name to signal that it wants to enter the critical section (`[n2w]`). When its name appears at the front of the queue, it is allowed to enter the critical section (rule `[w2c]`). When it has finished, it can go back to normal (rule `[c2n]`). Finally, a normal process may leave the protocol (`[exit]`).

The QLOCK Mutual Exclusion Protocol (II)

Processes are numbers. There is a left area for processes outside the protocol, and a protocol area (inside angle brackets). Processes outside can join the protocol ([join]). The protocol area has normal, waiting, and critical stages, plus a waiting queue, where a process can register its name to signal that it wants to enter the critical section ([n2w]). When its name appears at the front of the queue, it is allowed to enter the critical section (rule [w2c]). When it has finished, it can go back to normal (rule [c2n]). Finally, a normal process may leave the protocol ([exit]).

We can verify two important invariants of QLOCK, namey,

The QLOCK Mutual Exclusion Protocol (II)

Processes are numbers. There is a left area for processes outside the protocol, and a protocol area (inside angle brackets). Processes outside can join the protocol (`[join]`). The protocol area has normal, waiting, and critical stages, plus a waiting queue, where a process can register its name to signal that it wants to enter the critical section (`[n2w]`). When its name appears at the front of the queue, it is allowed to enter the critical section (rule `[w2c]`). When it has finished, it can go back to normal (rule `[c2n]`). Finally, a normal process may leave the protocol (`[exit]`).

We can verify two important invariants of QLOCK, namey,

- **Mutual Exclusion**, i.e., the critical section is either empty or has at most one process, and

The QLOCK Mutual Exclusion Protocol (II)

Processes are numbers. There is a left area for processes outside the protocol, and a protocol area (inside angle brackets). Processes outside can join the protocol (`[join]`). The protocol area has normal, waiting, and critical stages, plus a waiting queue, where a process can register its name to signal that it wants to enter the critical section (`[n2w]`). When its name appears at the front of the queue, it is allowed to enter the critical section (rule `[w2c]`). When it has finished, it can go back to normal (rule `[c2n]`). Finally, a normal process may leave the protocol (`[exit]`).

We can verify two important invariants of QLOCK, namey,

- **Mutual Exclusion**, i.e., the critical section is either empty or has at most one process, and
- **Deadlock Freedom**, i.e., the protocol never stops.

The QLOCK Mutual Exclusion Protocol (II)

Processes are numbers. There is a left area for processes outside the protocol, and a protocol area (inside angle brackets). Processes outside can join the protocol ([join]). The protocol area has normal, waiting, and critical stages, plus a waiting queue, where a process can register its name to signal that it wants to enter the critical section ([n2w]). When its name appears at the front of the queue, it is allowed to enter the critical section (rule [w2c]). When it has finished, it can go back to normal (rule [c2n]). Finally, a normal process may leave the protocol ([exit]).

We can verify two important invariants of QLOCK, namey,

- **Mutual Exclusion**, i.e., the critical section is either empty or has at most one process, and
- **Deadlock Freedom**, i.e., the protocol never stops.

The QLOCK Mutual Exclusion Protocol (II)

Processes are numbers. There is a left area for processes outside the protocol, and a protocol area (inside angle brackets). Processes outside can join the protocol (`[join]`). The protocol area has normal, waiting, and critical stages, plus a waiting queue, where a process can register its name to signal that it wants to enter the critical section (`[n2w]`). When its name appears at the front of the queue, it is allowed to enter the critical section (rule `[w2c]`). When it has finished, it can go back to normal (rule `[c2n]`). Finally, a normal process may leave the protocol (`[exit]`).

We can verify two important invariants of QLOCK, namey,

- **Mutual Exclusion**, i.e., the critical section is either empty or has at most one process, and
- **Deadlock Freedom**, i.e., the protocol never stops.

from, e.g., the initial state `init(7)` with seven processes.

Verifying Mutual Exclusion and Deadlock Freedom for QLOCK

We can specify the violation of mutual exclusion in QLOCK by the constructor pattern:

```
{S < U | W | C i j | Q >}
```

Verifying Mutual Exclusion and Deadlock Freedom for QLOCK

We can specify the violation of mutual exclusion in QLOCK by the constructor pattern:

```
{S < U | W | C i j | Q >}
```

Note that, by *ACU*, C could be mt.

Verifying Mutual Exclusion and Deadlock Freedom for QLOCK

We can specify the violation of mutual exclusion in QLOCK by the constructor pattern:

```
{S < U | W | C i j | Q >}
```

Note that, by *ACU*, C could be mt. We can then verify mutual exclusion with the search command:

Verifying Mutual Exclusion and Deadlock Freedom for QLOCK

We can specify the violation of mutual exclusion in QLOCK by the constructor pattern:

```
{S < U | W | C i j | Q >}
```

Note that, by *ACU*, C could be mt. We can then verify mutual exclusion with the search command:

```
Maude> search init(7) =>* {S < U | W | C i j | Q >} .

No solution.
```

Verifying Mutual Exclusion and Deadlock Freedom for QLOCK

We can specify the violation of mutual exclusion in QLOCK by the constructor pattern:

```
{S < U | W | C i j | Q >}
```

Note that, by *ACU*, C could be mt. We can then verify mutual exclusion with the search command:

```
Maude> search init(7) =>* {S < U | W | C i j | Q >} .

No solution.
```

Verifying deadlock freedom is even easier:

Verifying Mutual Exclusion and Deadlock Freedom for QLOCK

We can specify the violation of mutual exclusion in QLOCK by the constructor pattern:

```
{S < U | W | C i j | Q >}
```

Note that, by *ACU*, C could be mt. We can then verify mutual exclusion with the search command:

```
Maude> search init(7) =>* {S < U | W | C i j | Q >} .

No solution.
```

Verifying deadlock freedom is even easier:

```
Maude> search init(7) =>! X:State .

No solution.
```

Verifying Mutual Exclusion and Deadlock Freedom for QLOCK

We can specify the violation of mutual exclusion in QLOCK by the constructor pattern:

```
{S < U | W | C i j | Q >}
```

Note that, by *ACU*, C could be mt. We can then verify mutual exclusion with the search command:

```
Maude> search init(7) =>* {S < U | W | C i j | Q >} .

No solution.
```

Verifying deadlock freedom is even easier:

```
Maude> search init(7) =>! X:State .

No solution.
```

Bounded Model Checking of Invariants

Although explicit state search can be a quite effective model checking technique for invariants, it has some limitations:

Bounded Model Checking of Invariants

Although explicit state search can be a quite effective model checking technique for invariants, it has some limitations:

- if the set of reachable states is infinite and the invariant is satisfied, the search process never terminates;

Bounded Model Checking of Invariants

Although explicit state search can be a quite effective model checking technique for invariants, it has some limitations:

- if the set of reachable states is infinite and the invariant is satisfied, the search process never terminates;
- it can only explore a finite set of initial states (one at a time); but the set of initial states may be infinite (e.g, as in QLOCK);

Bounded Model Checking of Invariants

Although explicit state search can be a quite effective model checking technique for invariants, it has some limitations:

- if the set of reachable states is infinite and the invariant is satisfied, the search process never terminates;
- it can only explore a finite set of initial states (one at a time); but the set of initial states may be infinite (e.g, as in `QLOCK`);
- even if the number of reachable states is finite, it may be too large to be explored due to time and memory limitations.

Bounded Model Checking of Invariants

Although explicit state search can be a quite effective model checking technique for invariants, it has some limitations:

- if the set of reachable states is infinite and the invariant is satisfied, the search process never terminates;
- it can only explore a finite set of initial states (one at a time); but the set of initial states may be infinite (e.g, as in `QLOCK`);
- even if the number of reachable states is finite, it may be too large to be explored due to time and memory limitations.

There are several alternatives:

Bounded Model Checking of Invariants

Although explicit state search can be a quite effective model checking technique for invariants, it has some limitations:

- if the set of reachable states is infinite and the invariant is satisfied, the search process never terminates;
- it can only explore a finite set of initial states (one at a time); but the set of initial states may be infinite (e.g, as in `QLOCK`);
- even if the number of reachable states is finite, it may be too large to be explored due to time and memory limitations.

There are several alternatives: (1) Search states only up to a given depth bound.

Bounded Model Checking of Invariants

Although explicit state search can be a quite effective model checking technique for invariants, it has some limitations:

- if the set of reachable states is infinite and the invariant is satisfied, the search process never terminates;
- it can only explore a finite set of initial states (one at a time); but the set of initial states may be infinite (e.g, as in `QLOCK`);
- even if the number of reachable states is finite, it may be too large to be explored due to time and memory limitations.

There are several alternatives: (1) Search states only up to a given depth bound. (2) Explore an infinite set of states by symbolic model checking.

Bounded Model Checking of Invariants

Although explicit state search can be a quite effective model checking technique for invariants, it has some limitations:

- if the set of reachable states is infinite and the invariant is satisfied, the search process never terminates;
- it can only explore a finite set of initial states (one at a time); but the set of initial states may be infinite (e.g, as in `QLOCK`);
- even if the number of reachable states is finite, it may be too large to be explored due to time and memory limitations.

There are several alternatives: (1) Search states only up to a given depth bound. (2) Explore an infinite set of states by symbolic model checking. (3) Use an equational abstraction to make the set of reachable states finite.

Bounded Model Checking of Invariants

Although explicit state search can be a quite effective model checking technique for invariants, it has some limitations:

- if the set of reachable states is infinite and the invariant is satisfied, the search process never terminates;
- it can only explore a finite set of initial states (one at a time); but the set of initial states may be infinite (e.g, as in `QLOCK`);
- even if the number of reachable states is finite, it may be too large to be explored due to time and memory limitations.

There are several alternatives: (1) Search states only up to a given depth bound. (2) Explore an infinite set of states by symbolic model checking. (3) Use an equational abstraction to make the set of reachable states finite. (4) Use methods that combine symbolic model checking and theorem proving.

Bounded Model Checking of Invariants

Although explicit state search can be a quite effective model checking technique for invariants, it has some limitations:

- if the set of reachable states is infinite and the invariant is satisfied, the search process never terminates;
- it can only explore a finite set of initial states (one at a time); but the set of initial states may be infinite (e.g, as in `QLOCK`);
- even if the number of reachable states is finite, it may be too large to be explored due to time and memory limitations.

There are several alternatives: (1) Search states only up to a given depth bound. (2) Explore an infinite set of states by symbolic model checking. (3) Use an equational abstraction to make the set of reachable states finite. (4) Use methods that combine symbolic model checking and theorem proving. In this lecture I will explore alternative (1).

Bounded Model Checking of Invariants

Although explicit state search can be a quite effective model checking technique for invariants, it has some limitations:

- if the set of reachable states is infinite and the invariant is satisfied, the search process never terminates;
- it can only explore a finite set of initial states (one at a time); but the set of initial states may be infinite (e.g, as in `QLOCK`);
- even if the number of reachable states is finite, it may be too large to be explored due to time and memory limitations.

There are several alternatives: (1) Search states only up to a given depth bound. (2) Explore an infinite set of states by symbolic model checking. (3) Use an equational abstraction to make the set of reachable states finite. (4) Use methods that combine symbolic model checking and theorem proving. In this lecture I will explore alternative (1). Alternatives (2)–(3) will be explored later.

Bounded Model Checking of Invariants (II)

Bounded model checking is an appealing and widely used formal analysis method for two reasons:

Bounded Model Checking of Invariants (II)

Bounded model checking is an appealing and widely used formal analysis method for two reasons: (1) the number of reachable states may be infinite; or

Bounded Model Checking of Invariants (II)

Bounded model checking is an appealing and widely used formal analysis method for two reasons: (1) the number of reachable states may be infinite; or (2) it may be finite but too large (e.g., for a complex microprocessor design).

Bounded Model Checking of Invariants (II)

Bounded model checking is an appealing and widely used formal analysis method for two reasons: (1) the number of reachable states may be infinite; or (2) it may be finite but too large (e.g., for a complex microprocessor design). Bounded model checking cannot guarantee that an invariant holds everywhere; but it can either:

Bounded Model Checking of Invariants (II)

Bounded model checking is an appealing and widely used formal analysis method for two reasons: (1) the number of reachable states may be infinite; or (2) it may be finite but too large (e.g., for a complex microprocessor design). Bounded model checking cannot guarantee that an invariant holds everywhere; but it can either: (i) find very useful and often subtle counterexamples; or

Bounded Model Checking of Invariants (II)

Bounded model checking is an appealing and widely used formal analysis method for two reasons: (1) the number of reachable states may be infinite; or (2) it may be finite but too large (e.g., for a complex microprocessor design). Bounded model checking cannot guarantee that an invariant holds everywhere; but it can either: (i) find very useful and often subtle counterexamples; or (ii) guarantee that, up to a certain search depth, the invariant holds.

Bounded Model Checking of Invariants (II)

Bounded model checking is an appealing and widely used formal analysis method for two reasons: (1) the number of reachable states may be infinite; or (2) it may be finite but too large (e.g., for a complex microprocessor design). Bounded model checking cannot guarantee that an invariant holds everywhere; but it can either: (i) find very useful and often subtle counterexamples; or (ii) guarantee that, up to a certain search depth, the invariant holds.

Bounded model checking of invariants is supported by Maude's bounded depth breadth first search command.

Bounded Model Checking of Invariants (II)

Bounded model checking is an appealing and widely used formal analysis method for two reasons: (1) the number of reachable states may be infinite; or (2) it may be finite but too large (e.g., for a complex microprocessor design). Bounded model checking cannot guarantee that an invariant holds everywhere; but it can either: (i) find very useful and often subtle counterexamples; or (ii) guarantee that, up to a certain search depth, the invariant holds.

Bounded model checking of invariants is supported by Maude's bounded depth breadth first search command.

Consider the following specification of a readers-writers system.

Bounded Model Checking of Invariants (III)

```
mod R&W is
  protecting NAT .
  sort Config .
  op <_,_> : Nat Nat -> Config [ctor] .   --- readers/writers

  vars R W : Nat .

  rl < 0, 0 > => < 0, s(0) > .
  rl < R, s(W) > => < R, W > .
  rl < R, 0 > => < s(R), 0 > .
  rl < s(R), W > => < R, W > .
endm
```

Bounded Model Checking of Invariants (III)

```
mod R&W is
  protecting NAT .
  sort Config .
  op <_,_> : Nat Nat -> Config [ctor] .   --- readers/writers

  vars R W : Nat .

  rl < 0, 0 > => < 0, s(0) > .
  rl < R, s(W) > => < R, W > .
  rl < R, 0 > => < s(R), 0 > .
  rl < s(R), W > => < R, W > .
endm
```

A state is represented by a tuple `< R, W >` indicating the number `R` of readers and the number `W` of writers accessing a critical resource.

Bounded Model Checking of Invariants (III)

```
mod R&W is
  protecting NAT .
  sort Config .
  op <_,_> : Nat Nat -> Config [ctor] .  --- readers/writers

  vars R W : Nat .

  rl < 0, 0 > => < 0, s(0) > .
  rl < R, s(W) > => < R, W > .
  rl < R, 0 > => < s(R), 0 > .
  rl < s(R), W > => < R, W > .
endm
```

A state is represented by a tuple `< R, W >` indicating the number `R` of readers and the number `W` of writers accessing a critical resource. Readers and writers can leave the resource at any time;

Bounded Model Checking of Invariants (III)

```
mod R&W is
  protecting NAT .
  sort Config .
  op <_,_> : Nat Nat -> Config [ctor] .  --- readers/writers

  vars R W : Nat .

  rl < 0, 0 > => < 0, s(0) > .
  rl < R, s(W) > => < R, W > .
  rl < R, 0 > => < s(R), 0 > .
  rl < s(R), W > => < R, W > .
endm
```

A state is represented by a tuple `< R, W >` indicating the number `R` of readers and the number `W` of writers accessing a critical resource. Readers and writers can leave the resource at any time; but writers can only gain access to it if no other process is using it,

Bounded Model Checking of Invariants (III)

```
mod R&W is
  protecting NAT .
  sort Config .
  op <_,_> : Nat Nat -> Config [ctor] .  --- readers/writers

  vars R W : Nat .

  rl < 0, 0 > => < 0, s(0) > .
  rl < R, s(W) > => < R, W > .
  rl < R, 0 > => < s(R), 0 > .
  rl < s(R), W > => < R, W > .
endm
```

A state is represented by a tuple `< R, W >` indicating the number `R` of readers and the number `W` of writers accessing a critical resource. Readers and writers can leave the resource at any time; but writers can only gain access to it if no other process is using it, and readers only if there are no writers.

Bounded Model Checking of Invariants (IV)

From initial state `< 0, 0 >` we want to verify three invariants:

Bounded Model Checking of Invariants (IV)

From initial state `< 0, 0 >` we want to verify three invariants:

- mutual exclusion: readers and writers never access the resource simultaneously: only readers or only writers can do so at any given time.

Bounded Model Checking of Invariants (IV)

From initial state < 0, 0 > we want to verify three invariants:

- mutual exclusion: readers and writers never access the resource simultaneously: only readers or only writers can do so at any given time.
- one writer: at most one writer will be able to access the resource at any given time.

Bounded Model Checking of Invariants (IV)

From initial state `< 0, 0 >` we want to verify three invariants:

- mutual exclusion: readers and writers never access the resource simultaneously: only readers or only writers can do so at any given time.
- one writer: at most one writer will be able to access the resource at any given time.
- deadlock freedom: there are no deadlocks.

Bounded Model Checking of Invariants (IV)

From initial state `< 0, 0 >` we want to verify three invariants:

- mutual exclusion: readers and writers never access the resource simultaneously: only readers or only writers can do so at any given time.
- one writer: at most one writer will be able to access the resource at any given time.
- deadlock freedom: there are no deadlocks.

Violation of mutual exclusion can be specified with the pattern:

```
< s(N:Nat), s(M:Nat) >
```

Bounded Model Checking of Invariants (IV)

From initial state `< 0, 0 >` we want to verify three invariants:

- mutual exclusion: readers and writers never access the resource simultaneously: only readers or only writers can do so at any given time.
- one writer: at most one writer will be able to access the resource at any given time.
- deadlock freedom: there are no deadlocks.

Violation of mutual exclusion can be specified with the pattern:

```
< s(N:Nat), s(M:Nat) >
```

And violation of one writer with the pattern:

```
< N:Nat, s(s(M:Nat)) >
```

Bounded Model Checking of Invariants (IV)

From initial state `< 0, 0 >` we want to verify three invariants:

- mutual exclusion: readers and writers never access the resource simultaneously: only readers or only writers can do so at any given time.
- one writer: at most one writer will be able to access the resource at any given time.
- deadlock freedom: there are no deadlocks.

Violation of mutual exclusion can be specified with the pattern:

```
< s(N:Nat), s(M:Nat) >
```

And violation of one writer with the pattern:

```
< N:Nat, s(s(M:Nat)) >
```

However, since the number of readers can grow unboundedly, Maude's `search` commands to find counterexamples instantiating either of these two patterns from `< 0, 0 >` search forever.

Bounded Model Checking of Invariants (V)

We can however perform bounded model checking of these three invariants by giving a 10^6 depth bound:

Bounded Model Checking of Invariants (V)

We can however perform bounded model checking of these three invariants by giving a 10^6 depth bound:

```
Maude> search [1, 1000000] < 0,0 > =>* < s(N:Nat), s(M:Nat) > .
No solution.
states: 1000002  rewrites: 2000001 in 36480ms cpu (50317ms real)

Maude> search [1, 1000000] < 0,0 > =>* < N:Nat, s(s(M:Nat)) > .
No solution.
states: 1000002  rewrites: 2000001 in 38910ms cpu (41650ms real)

Maude> search [1, 1000000] < 0,0 > =>! C:Config .
No solution.
states: 1000003  rewrites: 2000002 in 5752ms cpu (5821ms real)
```

Bounded Model Checking of Invariants (V)

We can however perform bounded model checking of these three invariants by giving a 10^6 depth bound:

```
Maude> search [1, 1000000] < 0,0 > =>* < s(N:Nat), s(M:Nat) > .
No solution.
states: 1000002  rewrites: 2000001 in 36480ms cpu (50317ms real)

Maude> search [1, 1000000] < 0,0 > =>* < N:Nat, s(s(M:Nat)) > .
No solution.
states: 1000002  rewrites: 2000001 in 38910ms cpu (41650ms real)

Maude> search [1, 1000000] < 0,0 > =>! C:Config .
No solution.
states: 1000003  rewrites: 2000002 in 5752ms cpu (5821ms real)
```

Thus verifying these three invariants up to depth 10^6.