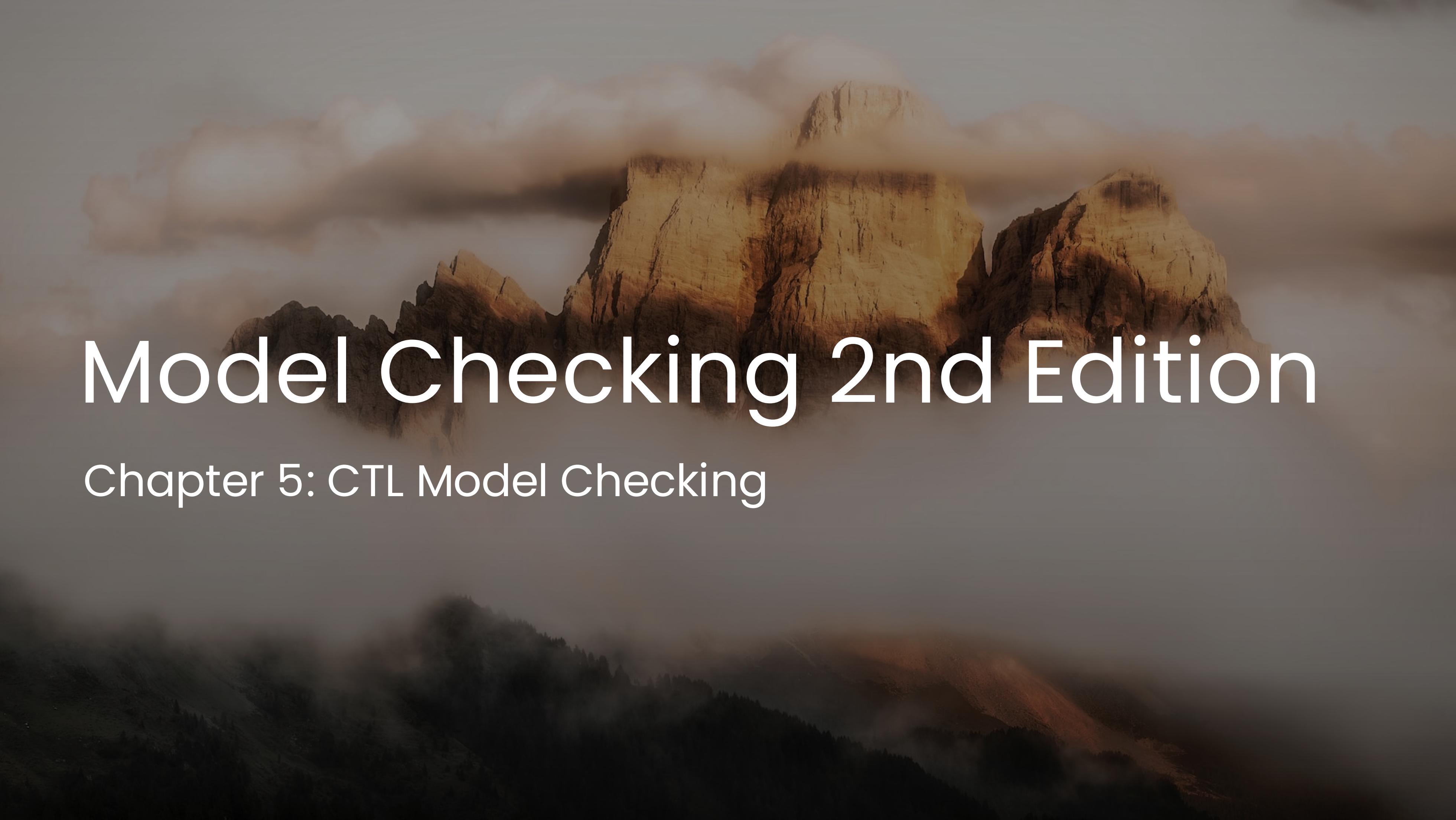


Model Checking 2nd Edition

Chapter 5: CTL Model Checking

CTL Model Checking

The model-checking problem and algorithms for CTL

Wang Qirui 1W202047
5/22/2023

Table of Contents

Introduction

What is CTL model checking?

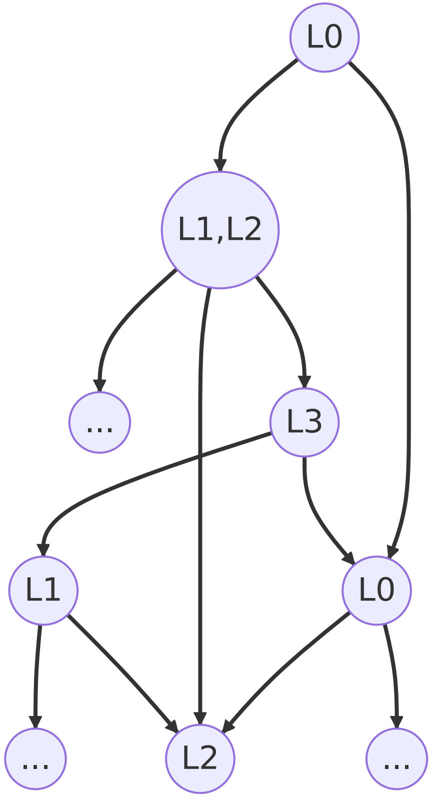
Is $M = (AP, S, R, S_0, L)$ a *model* of CTL formula f (i.e., $M \models f$)?

Find the set $[\![f]\!]_M$ of all states in M that satisfy f:

$$[\![f]\!]_M \models \{s \in S \mid M, s \models f\}$$

- Problem can be solved by checking $S_0 \subseteq [\![f]\!]$.
  - S_0 is not needed during model checking.
- We fix AP for the rest of the chapter.
  - AP is not needed too.

Represented as a directed graph (S, R) with labeling L.



Explicit-State CTL Model Checking

A model-checking algorithm for CTL

Explicit-State CTL Model Checking

Introduction

How to determine which states in S satisfy f?

Labeling each state s with $label(s)$, which is a set of subformulas that are true in s.

1. Let $label(s)$ be $L(S)$.
2. Then a sub-routine with multiple stages
    1. During the i-th stage, subformulas with $i-1$ nested CTL operators are processed.
    2. Subformula that is processed is added to the labeling of each state that in which it is true.
3. Terminated with $M, s \models f \iff f \in label(s)$.

Explicit-State CTL Model Checking

Introduction

As mentioned in 4.3.1, any CTL formula can be expressed in terms of

$$\neg, \vee, \mathbf{EX}, \mathbf{EU}, \mathbf{EG}.$$

The intermediate stages of the algorithm is able to handle *6* cases:

- g is atomic
- g has one of the forms: $\neg f_1$, $f_1 \vee f_2$, $\mathbf{EX} f_1$, $\mathbf{E}(f_1 \mathbf{U} f_2)$, $\mathbf{EG} f_1$.

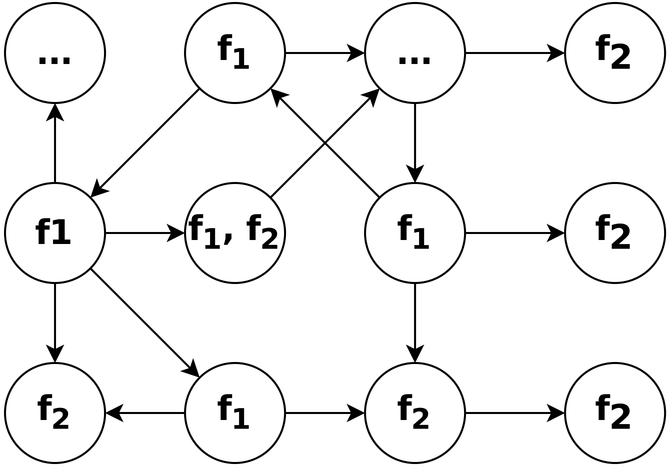
For example:

- $\neg f_1$: states that are not labeled by f_1.
- $f_1 \vee f_2$: states that is labeled by f_1 or f_2.
- $\mathbf{EX} f_1$: states that has some successor labeled by f_1.

Example

Handling formula of form $g = \mathbf{E}(f_1 \mathbf{U} f_2)$

1. Find all states that are labeled with f_2.
2. Search backward using converse of R for all states that can be reached by a path that all states are labeled with f_1.
3. All such states are labeled with g.



Example-1.1

procedure *CheckEU*(f_1, f_2)
  $T := \{\, s \mid f_2 \in label(s) \,\}$;
  for all $s \in T$ **do** $label(s) := label(s) \cup \{\, \mathbf{E}(f_1 \,\mathbf{U}\, f_2) \,\}$;
  while $T \neq \emptyset$ **do**
    choose $s \in T$;
    $T := T \setminus \{s\}$;
    for all t **such that** $R(t, s)$ **do**
      if $\mathbf{E}(f_1 \,\mathbf{U}\, f_2) \notin label(t)$ **and** $f_1 \in label(t)$ **then**
        $label(t) := label(t) \cup \{\, \mathbf{E}(f_1 \,\mathbf{U}\, f_2) \,\}$;
        $T := T \cup \{t\}$;
      end if
    end for all
  end while
end procedure

Figure 5.1

Procedure for labeling the states satisfying $g = \mathbf{E}(f_1 \mathbf{U} f_2)$

Another Example

The case of $g = \mathbf{EG} f_1$

Based on the decomposition of the graph into *nontrivial strongly connected components*.

> 1. **Strongly connected component (SCC)** is a subgraph such that every node is reachable from every other node along a directed path.
> 2. **Maximal SCC (MSCC)** is a SCC that is not a subset of any other SCC.
> 3. **Nontrivial SCC** is a SCC with more than one node or with only one node with a self-loop.

Let's retain only states that satisfy f_1:

$$\begin{aligned} M' &= (S', R', L') \\ S' &= \{s \in S \mid M, s \models f_1\} \\ R' &\models R|_{S' \times S'} \\ L' &\models L|_{S'} \end{aligned}$$

Note: R' may not be left-total in this case.

Another Example

The case of $g = \mathbf{EG}\, f_1$

Lemma 5.1

$M,\ s \models \mathbf{EG}\, f_1$ if and only if the following conditions are satisfied:

1. $s \in S'$.
2. There exists a path in M' that leads from s to some node t in a nontrivial MSCC C of the graph (S', R').

Proof.

- **Sufficiency**

It's clearly that:

$$M,\ s \models \mathbf{EG}\, f_1 \implies s \in S'.$$

Another Example

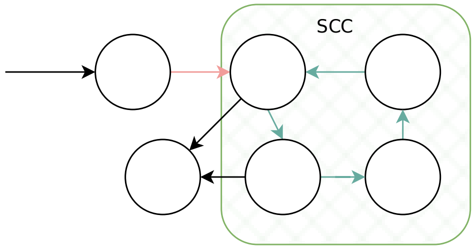
Proof to Lemma 5.1

Let π be an infinite path in M starting at s, then:

$$M,\ \pi \models \mathbf{G} f_1 \implies \pi \in S'.$$

Since M is finite, it's possible to write π as $\pi = \pi_0 \pi_1$, where π_0 is a finite initial segment and π_1 is an infinite suffix of π. Then we will show that C is a nontrivial SCC.

Let C be the set of all states in π_1. and take states $s_1, s_2 \in C$, since π_1 is an infinite path, then the segment from s_1 to s_2 is a finite path from s_1 to s_2 within C. So C is a nontrivial SCC.



Note: if C is not maximal, then it's contained in an MSCC C' and π_0 leads to C' since it leads to C.

Another Example

Proof to Lemma 5.1

- **Necessity**

Let π_0 be the path from s to t in M' and π_1 be the path of length at least 1 from t back to t.

- Since t is in a nontrivial MSCC, then π_1's existence is guaranteed.
- All the states on the infinite path $\pi = \pi_0\pi_1^\omega$ satisfy f_1.

Since π is a path from s in M, then $M,\ s \models \mathbf{EG} f_1$. $\square$

Lemma 5.1 shows that the search of infinite paths can be reduced to the search of an MSCC.

- The search of SCCs is exponential. (they might include all subsets of S)
- The search of MSCCs can be done in linear time.

Another Example

The case of $g = \mathbf{EG}\, f_1$

Then the algorithm for the case of $g = \mathbf{EG}\, f_1$, with the help of Lemma 5.1, turns out to be:

1. Construct the Kripke structure $M' = (S', R', L')$.
2. Partition the graph (S', R') into its MSCCs using Tarjan's algorithm[1].
3. Find those states belonging to nontrivial ones.
4. Work backward using the converse of R' like the case of $\mathbf{E}(f_1 \,\mathbf{U}\, f_2)$.

procedure *CheckEG*(f_1)
  $S' := \{\, s \mid f_1 \in \mathit{label}(s) \,\}$;
  $\mathit{MSCC} := \{\, C \mid C$ is a nontrivial maximal SCC of $S' \,\}$;
  $T := \bigcup_{C \in \mathit{MSCC}} \{\, s \mid s \in C \,\}$;
  for all $s \in T$ **do** $\mathit{label}(s) := \mathit{label}(s) \cup \{\, \mathbf{EG}\, f_1 \,\}$;
  while $T \neq \emptyset$ **do**
    choose $s \in T$;
    $T := T \setminus \{s\}$;
    for all t **such that** $t \in S'$ **and** $R(t, s)$ **do**
      if $\mathbf{EG}\, f_1 \notin \mathit{label}(t)$ **then**
        $\mathit{label}(t) := \mathit{label}(t) \cup \{\, \mathbf{EG}\, f_1 \,\}$;
        $T := T \cup \{t\}$;
      end if
    end for all
  end while
end procedure

Figure 5.2

Procedure for labeling the states satisfying $g = \mathbf{EG}\, f_1$

[1] Tarjan's algorithm (will be introduced in 5.5) finds the set of all MSCCs with time complexity $O(|S'| + |R'|)$.

Explicit-State CTL Model Checking

General approach

How to handle arbitrary CTL formula f?

- Decompose the formula into subformulas and apply the state-labeling algorithm to them.
- Start with the **shortest** and **most deeply nested** subformulas, and **work outward**.
  - All subformulas of formula currently processing are guaranteed to be processed.

Since each pass takes time $O(|S| + |R|)$ and f has at most $|f|$ different subformulas, the total pass requires time:

$$O(|S| + |R|) \cdot |f| = O(|f| \cdot (|S| + |R|)).$$

Theorem 5.2

There is an algorithm for determining $[\![f]\!]$ that runs in time $O(|f| \cdot (|S| + |R|))$.

Explicit-State CTL Model Checking

General approach

- Theorem 5.2 holds for every CTL formula over $\mathbf{EX}$, $\mathbf{E}(\mathbf{U})$, and $\mathbf{EG}$.
  - Every other CTL formula can be expressed by means of these three operators. (Chapter 4)
  - Preprocess the CTL formula to obtain a formula containing only $\mathbf{EX}$, $\mathbf{E}(\mathbf{U})$, and $\mathbf{EG}$.
    - All translations are linear in the size of the original formula, except for $\mathbf{A}(\mathbf{U})$.

Recall that:

$$\mathbf{A}(f\mathbf{U}g) \equiv \neg\mathbf{E}(\neg g\mathbf{U}(\neg f \wedge \neg g)) \wedge \neg\mathbf{EG}\neg g.$$

There are only 8 different subformulas, so the overall time complexity is **preserved**.

Example: Microwave oven

Model-checking algorithm for CTL

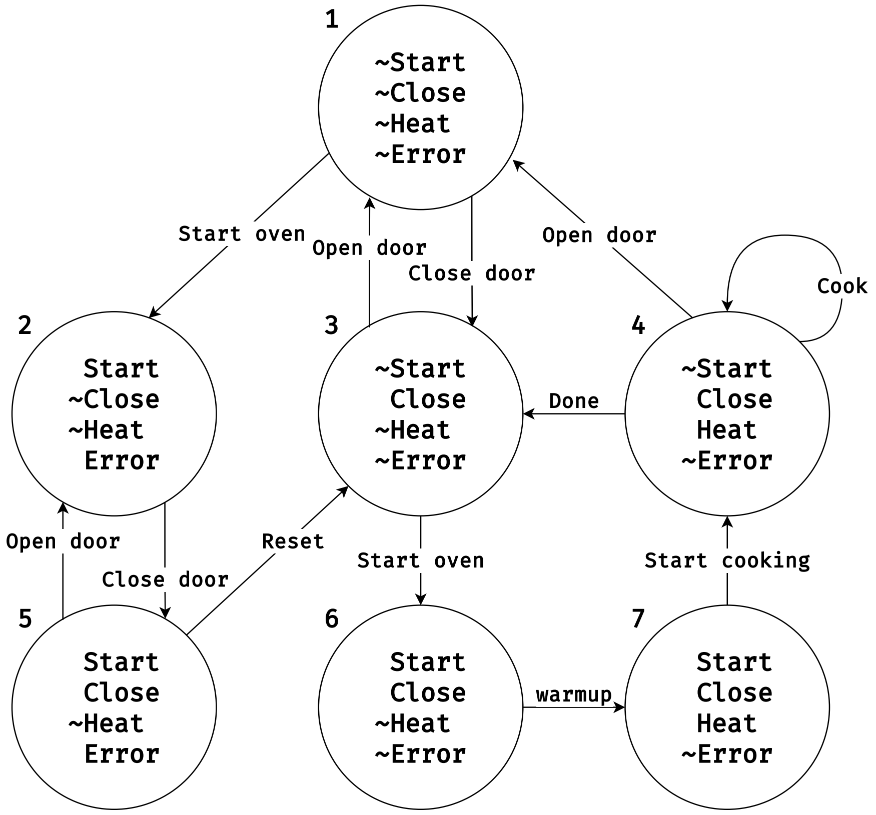
Let's check the following CTL formula:

$$\mathbf{AG}(\mathit{Start} \to \mathbf{AF}\,\mathit{Heat}) \\ \equiv \neg\mathbf{EF}(\mathit{Start} \wedge \mathbf{EG}\neg\mathit{Heat}).$$

We start by computing the set of states satisfying the APs:

$$[\![\mathit{Start}]\!] = \{2, 5, 6, 7\}$$
$$[\![\mathit{Heat}]\!] = \{4, 7\}$$

To compute $[\![\mathbf{EG}\neg\mathit{Heat}]\!]$, we first find the set of nontrivial SCC in $S' = [\![\neg\mathit{Heat}]\!]$.



We have $MSCC = \{\{1, 2, 3, 5\}\}$.

Then, we set the set of states that should be labeled by $\mathbf{EG}\neg Heat$ to:

$$T = \bigcup_{C \in MSCC} \{s | s \in C\} = \{1, 2, 3, 5\}.$$

No other states in S' can reach state in T along a path **in** S', so the computation terminates with:

$$[\![\mathbf{EG}\neg Heat]\!] = \{1, 2, 3, 5\}.$$

Therefore:

$$[\![Start \land \mathbf{EG}\neg Heat]\!] = \{2, 5\}.$$

For $[\![\mathbf{EF}(Start \land \mathbf{EG}\neg Heat)]\!]$, we set

$$T = [\![Start \land \mathbf{EG}\neg Heat]\!].$$

And find states that can reach T:

$$[\![\mathbf{EF}(Start \land \mathbf{EG}\neg Heat)]\!] = \{1, 2, 3, 4, 5, 6, 7\}.$$

Finally, we have:

$$[\![\neg\mathbf{EF}(Start \land \mathbf{EG}\neg Heat)]\!] = \emptyset.$$

The initial state 1 is not in the set, so the system described by the Kripke structure **does not satisfy the given specification**.

Model-Checking CTL with Fairness Constraints

Extend the CTL model-checking algorithm to handle fairness constraints.

Model-Checking CTL with Fairness Constraints

Introduction

Let $M = (S, R, L, F)$ be a fair Kripke structure, and $F = \{P_1, \ldots, P_k\}$ be the set of fairness constraints.

A SCC C of the graph of M is fair with respect to F if and only if $\forall P_i \in F, \exists t_i \in (C \cap P_i)$.

We will give an algorithm for checking $\mathbf{EG} f_1$ with respect to a fair structure. To ensure that the algorithm is correct, a lemma similar to **Lemma 5.1** is required.

Similarly, we obtain a set M' from M by removing from S all states that f_1 does not fairly hold in.

$$
\begin{aligned}
M' &= (S', R', L', F') \\
S' &= \{s \in S \mid M, s \models_F f_1\} \\
R' &= R|_{S' \times S'} \\
L' &= L|_{S'} \\
F' &= \{P_i \cap S' \mid P_i \in F\}
\end{aligned}
$$

Model-Checking CTL with Fairness Constraints

Describe *CheckFairEG* algorithm

Lemma 5.3

$M,\ s \models_F \mathbf{E}_f\mathbf{G}f_1$ if and only if the following conditions are satisfied:

1. $s \in S'$.
2. There exists a path in S' that leads from s to some node t in a nontrivial maximal strongly connected component of C of the graph (S', R').

The proof of this lemma is also similar, and we omit it here.

For `CheckFairEG`(f_1), who adds $\mathbf{E}_\mathrm{f}\mathbf{G}f_1$ to the label of s for every s such that $M, s \models_F \mathbf{EG}f_1$, we assume likewise that:

$$f_1 \in label(s) \iff M, s \models_F f_1.$$

Model-Checking CTL with Fairness Constraints

Describe *CheckFairEG* algorithm

The procedure body is the same as `CheckEG` shown in **Figure 5.1**, except that *MSCC* now consists of the set of nontrivial **fair** MSCCs.

- The **complexity** of the algorithm is $O((|S| + |R|) \cdot |F|)$
  - It's necessary to determine which components are fair
  - Involves examining every component to see if it has a state from each fairness constraint.

How to generalize the algorithm to check other CTL formulas?

- Introduce an additional **atomic proposition** $fair$.
  - Holds $true$ at state s if and only if there is a fair path starting from s, so,

$$fair \equiv \mathbf{E}_{\mathrm{f}}\mathbf{G}\, true$$

  - Procedure `CheckFairEG` can be used to label states with $fair$.

Model-Checking CTL with Fairness Constraints

Generalization of the algorithm

- To determine if $M, s \models_F \mathbf{E}_\mathrm{f}\mathbf{X} f_1$, check $M, s \models \mathbf{EX}(f_1 \wedge \mathit{fair})$
- To determine if $M, s \models_F \mathbf{E}_\mathrm{f}(f_1 \mathbf{U} f_2)$, check $M, s \models \mathbf{E}(f_1 \mathbf{U}(f_2 \wedge \mathit{fair}))$
  - By calling `CheckEU`$(f_1, f_2 \wedge \mathit{fair})$

Also, the total time complexity of the algorithm is $O(|f| \cdot (|S| + |R|) \cdot |F|)$.

Theorem 5.4

There is an algorithm for determining whether a CTL formula f is true with respect to the fair semantics in a state s of the structure $M = (S, R, L, F)$ that runs in time $O(|f| \cdot (|S| + |R|) \cdot |F|)$

Similarly, this theorem's correctness can be proved like **Theorem 5.2**.

Example: Microwave Oven

Illustrating the use of fairness constraints

We check the formula below with the same model as **Figure 5.3**:

$$\mathbf{A}_{\mathrm{f}}\mathbf{G}(\mathit{Start} \to \mathbf{A}_{\mathrm{f}}\mathbf{F}\mathit{Heat}) \equiv \neg\mathbf{E}_{\mathrm{f}}\mathbf{F}(\mathit{Start} \land \mathbf{E}_{\mathrm{f}}\mathbf{G}\neg\mathit{Heat})$$

We only look at the paths where **the user always uses the microwave oven correctly** and keep other variables unchanged:

$$F = \{P\}, \quad P = \{s \mid s \models \mathit{Start} \land \mathit{Close} \land \neg\mathit{Error}\}.$$

For $S' = [\![\neg\mathit{Heat}]\!]$, we have $\mathit{MSCC} = \{1, 2, 3, 5\}$, which is *not fair*, since no state in MSCC satisfies $\mathit{Start} \land \mathit{Close} \land \neg\mathit{Error}$.

So we have $[\![\mathbf{E}_{\mathrm{f}}\mathbf{G}\neg\mathit{Heat}]\!] = [\![\mathbf{E}_{\mathrm{f}}\mathbf{F}(\mathit{Start} \land \mathbf{E}_{\mathrm{f}}\mathbf{G}\neg\mathit{Heat})]\!] = \emptyset$.

Finally, we have $[\![\neg(\mathbf{E}_{\mathrm{f}}\mathbf{F}(\mathit{Start} \land \mathbf{E}_{\mathrm{f}}\mathbf{G}\neg\mathit{Heat}))]\!] = \{1, 2, 3, 4, 5, 6, 7\}$. So all states of the program satisfy the formula under the given fairness constraints.

CTL Model Checking via Fixpoint Computation

Algorithm that manipulates entire sets

CTL Model Checking via Fixpoint Computation

Introduction

In section 5.1, we discussed the explicit model-checking algorithm, which manipulates individual states and transitions.

Symbolic model-checking algorithms manipulate entire sets of states and transitions.

- Based on *fixpoint* characterization of the temporal logic operators.
- Use ordered binary decision diagrams (OBDDs) to represent sets of states and transitions.
- Quadratic time complexity, while explicit model-checking is linear.
- Significantly reduced space complexity.
  - Enables verification of systems with very large state spaces.

CTL Model Checking via Fixpoint Computation

Background on Fixpoint Theory

Let $M = (S, R, L)$ be a finite Kripke structure. The power set $\mathcal{P}(S)$ is a lattice under the inclusion order.

> A poset $(L, \leq)$ is called a **lattice** if for every pair of elements $a, b \in L$, there exists a **least upper bound** $a \vee b$ and a **greatest lower bound** $a \wedge b$.

- Each element S' of the lattice can be viewed as a *predicate* on S, where the predicate is true for a state s if and only if $s \in S'$.
- The least element of the lattice is the empty set $\emptyset$,we denote it by $false$.
- The greatest element of the lattice is the set S itself, we denote it by $true$.

Definition

A function that maps $\mathcal{P}(S)$ to $\mathcal{P}(S)$ is called a **predicate transformer**.

CTL Model Checking via Fixpoint Computation

Background on Fixpoint Theory

Definition

A set $S' \subseteq S$ is a **fixpoint** of a function $\tau : \mathcal{P}(S) \to \mathcal{P}(S)$ if $\tau(S') = S'$.

Let $\tau : \mathcal{P}(S) \to \mathcal{P}(S)$ be a predicate transformer, then:

1. τ is ***monotonic*** if $P \subseteq Q \implies \tau(P) \subseteq \tau(Q)$;
2. τ is $\cup$***-continuous*** if $P_1 \subseteq P_2 \subseteq \ldots \implies \tau(\cup_i P_i) = \cup_i \tau(P_i)$;
3. τ is $\cap$***-continuous*** if $P_1 \supseteq P_2 \supseteq \ldots \implies \tau(\cap_i P_i) = \cap_i \tau(P_i)$.

We write $\tau^i(Z)$ to denote i applications of τ to Z. That is,

$$\tau^0(Z) = Z$$
$$\tau^{i+1}(Z) = \tau(\tau^i(Z))$$

CTL Model Checking via Fixpoint Computation

Background on Fixpoint Theory

Theorem 5.5 (Tarski-Knaster)

Let τ be a predicate transformer on $\mathcal{P}(S)$. Then if τ is monotonic it has a greatest fixpoint, $\nu Z.\tau(Z)$, and a least fixpoint, $\mu Z.\tau(Z)$, defined as follows:

- $\nu Z.\tau(Z) = \bigcup\{Z \mid Z \subseteq \tau(Z)\}$.
- $\mu Z.\tau(Z) = \bigcap\{Z \mid Z \supseteq \tau(Z)\}$.

Furhtermore, if τ is $\cap$-continuous, then $\nu Z.\tau(Z) = \bigcap \tau^i(\mathit{true})$, and if τ is $\cup$-continuous, then $\mu Z.\tau(Z) = \bigcup \tau^i(\mathit{false})$.

Proof. Let $\Gamma = \{Z \mid Z \subseteq \tau(Z)\}$ and $P = \cup\Gamma$. Then $\forall Z \in \Gamma, Z \subseteq P$. Thus, for $Z \in \Gamma$:

- $Z \subseteq \tau(Z) \implies \tau(Z) \subseteq \tau(\tau(Z)) \implies \tau(Z) \in \Gamma$; (monotonicity)
- $\tau(Z) \subseteq \tau(P) \implies Z \subseteq \tau(Z) \subseteq \tau(P) \implies P \subseteq \tau(P)$;
- Then we have $P \in \Gamma$ and $\tau(P) \in \Gamma$.

CTL Model Checking via Fixpoint Computation

Proof of Theorem 5.5

By definition of Γ, $P \subseteq \Gamma$ and $\tau(P) \subseteq \Gamma$. And since $P = \cup\Gamma$, we have $\tau(P) \subseteq P$. Thus,

$$\tau(P) = P.$$

- Since $\subseteq$ is reflexive (for any set S, $S \subseteq S$), then every fixpoint of τ is also in Γ.
- As P includes all sets in Γ, then P must be the **greatest** fixpoint of τ.

For the second part of the theorem, we have:

$$\tau(S) \subseteq S \implies \tau(\tau(S)) \subseteq \tau(S) \implies \tau^{i+1}(S) \subseteq \tau^{i}(S)$$

And by coninuity, we have:

$$\tau(\cap\tau^{i}(S)) = \cap\tau^{i+1}(S) \supseteq \cap\tau^{i}(S) \implies \cap\tau^{i}(S) \in \Gamma \implies \cap\tau^{i}(S) \subseteq P$$

CTL Model Checking via Fixpoint Computation

Proof of Theorem 5.5

It's obvious that $P \subseteq S$, so we have:

$$P = \tau(P) \subseteq \tau(S) \subseteq S \implies P \subseteq \tau^i(S) \implies P \subseteq \cap \tau^i(S).$$

Therefore, $P = \cap \tau^i(S)$, that is, $\nu Z.\tau(Z) = \bigcap \tau^i(true)$. The proof for the least fixpoint is similar so it's omitted here. $\square$

The Knaster Tarski theorem implies:

- if τ is continuous, then it can be computed by a (possibly infinite) sequence of applications of τ .

Next, some lemmas will be proved to show that:

- for $\tau : \mathcal{P}(S) \to \mathcal{P}(S)$, if S is finite, then whenever τ is monotonic, it is also continuous.
- In this case, only a finite number of applications are needed.
- Finally, we obtain an algorithm for computing the fixpoints.

CTL Model Checking via Fixpoint Computation

Background on Fixpoint Theory

Lemma 5.6

If S is finite and τ is monotonic, then τ is also $\cap$-continuous and $\cup$-continuous.

Proof. Let $P_1 \subseteq P_2 \subseteq \ldots$ be a sequence of subsets of S, then we have $\tau(P_1) \subseteq \tau(P_2) \subseteq \ldots$

Since S is finite,

$$\exists j_0, (\forall j \geq j_0, P_j = P_{j_0}) \wedge (\forall j < j_0, P_j \subseteq P_{j_0}) \implies \cup_i P_i = P_{j_0} \implies \tau(\cup_i P_i) = \tau(P_{j_0}).$$

Also we have:

$$\exists j_0, (\forall j \geq j_0, \tau(P_j) = \tau(P_{j_0})) \wedge (\forall j < j_0, \tau(P_j) \subseteq \tau(P_{j_0})) \implies \cup_i \tau(P_i) = \tau(P_{j_0}).$$

Therefore τ is $\cup$-continuous. The proof that τ is $\cap$-continuous is similar, so omitted here. $\square$

Lemma 5.7

If τ is monotonic, then for every i, $\tau^i(\mathit{false}) \subseteq \tau^{i+1}(\mathit{false})$ and $\tau^i(\mathit{true}) \supseteq \tau^{i+1}(\mathit{true})$.

Recall that:

$$S \supseteq \tau(S) \implies \tau(S) \supseteq \tau(\tau(S)) \implies \tau^i(S) \supseteq \tau^{i+1}(S)$$
$$\emptyset \subseteq \tau(\emptyset) \implies \tau(\emptyset) \subseteq \tau(\tau(\emptyset)) \implies \tau^i(\emptyset) \subseteq \tau^{i+1}(\emptyset)$$

Lemma 5.8

If τ is monotonic and S is finite, then there is an integer i_0 such that for every $j \geq i_0$, $\tau^j(\mathit{false}) = \tau^{i_0}(\mathit{false})$. Similarly, there is an integer j_0 such that for every $j \geq j_0$, $\tau^j(\mathit{true}) = \tau^{j_0}(\mathit{true})$.

Lemma 5.9

If τ is monotonic and S is finite, then there is an integer i_0 such that $\mu Z.\tau(Z) = \tau^{i_0}(\mathit{false})$. Similarly, there is an integer j_0 such that $\nu Z.\tau(Z) = \tau^{j_0}(\mathit{true})$.

CTL Model Checking via Fixpoint Computation

Background on Fixpoint Theory

With these lemmas, we can compute the least fixpoint of τ if τ is monotonic.

And the **invariant** of the while loop is given by assertion:

$$(Q' = \tau(Q)) \wedge (Q' \subseteq \mu Z.\tau(Z)).$$

We can see that at the i-th iteration of the loop, $Q = \tau_{i-1}(\mathit{false})$ and $Q' = \tau_i(\mathit{false})$. Lemma 5.7 implies that

$$\mathit{false} \subseteq \tau(\mathit{false}) \subseteq \tau^2(\mathit{false}) \subseteq \ldots$$

```
Set Lfp(std::function<Set(Set&)> const &tau)
{
  Set Q{false};
  Set Q_p{tau(Q)};
  while (Q != Q_p) {
    Q = Q_p;
    Q_p = tau(Q_p);
  }
  return Q;
}
```

Procedure to compute the least fixpoint of τ.

CTL Model Checking via Fixpoint Computation

Background on Fixpoint Theory

Therefore, the maximum number of iterations of the loop is bounded by $|S|$. When the loop terminates, we have $Q = \tau(Q)$ and hence $Q = \mu Z.\tau(Z)$.

Similarly, we can compute the greatest fixpoint also in a finite number of iterations, and it returns $\nu Z.\tau(Z)$:

```
Set Gfp(std::function<Set(Set&)> const &tau)
{
  Set Q{true};
  Set Q_p{tau(Q)};
  while (Q != Q_p) {
    Q = Q_p;
    Q_p = tau(Q_p);
  }
  return Q;
}
```

Problem

Problem 5.1

Disjointness of MSCCs

Let C_1 and C_2 be two MSCCs. Prove that they are disjoint. Conclude that the sum of states over all MSCCs of M is bounded by the size of S.

Proof. Assume that $C_1 \cap C_2 \neq \emptyset$. Let $C_3 = C_1 \cup C_2$. Then any state that is reachable from C_3 is also reachable from either C_1 or C_2.

Therefore, C_3 is an MSCC that is strictly larger than both C_1 and C_2, which contradicts the assumption that C_1 and C_2 are maximal. Hence, $C_1 \cap C_2 = \emptyset$.

Since MSCCs are disjoint to each other, and the union of all MSCCs is S, the sum of states over all MSCCs of M is bounded by the size of S. □