NSWI101: System Behaviour Models and Verification

4. Computational Tree Logic

Jan Kofroň
Computational Tree Logic (CTL)
CTL model checking
Comparison of CTL and LTL
MODEL CHECKING

AG (start → AF heat)

Property specification
MODEL CHECKING

System model

CTL

AG (start → AF heat)

Property specification

Model Checking

Property satisfied

Property violated
Another temporal logic, differing from LTL in expressive power

*Computational tree* refers to ability to properties of computational subtrees (branching)

- as opposed to LTL that considers particular paths in isolation

The semantic model similar to LTL – also defined upon infinite paths of Kripke structure
Let AP be set of atomic propositions (Boolean variables).

CTL formulae are finite expressions created by following rules:

- $\top, \bot, p \in AP$ are CTL formulae

If $\varphi, \psi$ are CTL formulae, then the following are also CTL formulae:

- $\text{AX} \varphi$
- $\text{AF} \varphi$
- $\text{AG} \varphi$
- $A[\varphi \text{ U } \psi]$
- $\text{EX} \varphi$
- $\text{EF} \varphi$
- $\text{EG} \varphi$
- $E[\varphi \text{ U } \psi]$

Operators X, F, G, U have similar meaning as in LTL
Quantifiers A, E refer to paths – “all paths” vs. “exists a path”
Let $M = (S, R, L)$ be Kripke structure

- $\langle s \rightarrow t \rangle$ denotes transition from state $s$ to state $t$
- $\langle s_0 \rightarrow \cdots \rangle$ denotes infinite path $\langle s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \rangle$ starting at state $s_0$
CTL SEMANTICS

\[(M, s \models T) \land (M, s \not\models \bot)\]
\[(M, s \models p) \iff (p \in L(s))\]
\[(M, s \models \neg \varphi) \iff (M, s \not\models \varphi)\]
\[(M, s \models \varphi_1 \lor \varphi_2) \iff ((M, s \models \varphi_1) \lor (M, s \models \varphi_2))\]
\[(M, s \models AX \varphi) \iff (\forall \langle s \rightarrow t \rangle (M, t \models \varphi))\]
\[(M, s \models EX \varphi) \iff (\exists \langle s \rightarrow t \rangle (M, t \models \varphi))\]
\[(M, s \models AG \varphi) \iff (\forall \langle s_0 \rightarrow \rangle \forall i \geq 0 : M, s_i \models \varphi)\]
\[(M, s \models EG \varphi) \iff (\exists \langle s_0 \rightarrow \rangle \forall i \geq 0 : M, s_i \models \varphi)\]
\[(M, s \models AF \varphi) \iff (\forall \langle s_0 \rightarrow \rangle \exists i \geq 0 : M, s_i \models \varphi)\]
\[(M, s \models EF \varphi) \iff (\exists \langle s_0 \rightarrow \rangle \exists i \geq 0 : M, s_i \models \varphi)\]
\[(M, s \models A[\varphi_1 U \varphi_2]) \iff (\forall \langle s_0 \rightarrow \rangle \exists i \geq 0 : (M, s_i \models \varphi_2) \land (\forall (j < i)(M, s_j) \models \varphi_1))\]
\[(M, s \models E[\varphi_1 U \varphi_2]) \iff (\exists \langle s_0 \rightarrow \rangle \exists i \geq 0 : (M, s_i \models \varphi_2) \land (\forall (j < i)(M, s_j) \models \varphi_1))\]
CTL MODEL CHECKING

Based on identifying states of model satisfying sub-formulae of property formula:

1. Create derivation tree of property formula.
2. In bottom-up manner identify all states of model satisfying sub-formula associated with each node of derivation tree.
CTL MODEL CHECKING

Based on identifying states of model satisfying sub-formulae of property formula:
1. Create derivation tree of property formula.
2. In bottom-up manner identify all states of model satisfying sub-formula associated with each node of derivation tree.

\[
\text{EG (E}[p U q]\text{) } \wedge \neg (\text{EX } r)
\]

EG (E[p U q])

\[
\text{E[p U q]}
\]

\[
p \quad q
\]

\[
\neg (\text{EX } r)
\]

\[
\text{EX } r
\]

\[
r
\]
CTL MODEL CHECKING

CTL formula can be transformed to contain just $\neg$, $\land$, EG, EX, and EU operators

Various algorithms for identification of states satisfying particular sub-formulae exist

- explicit model checking – explicit representation of each state in memory
- symbolic model checking – representing sets of states by Boolean formulae
Identification of states satisfying particular sub-formulae:
- operators $\neg$, $\land$, and EX are trivial
- operators EG and EU require more complex algorithms
Explicit CTJ model checking: EU operator

function CHECKEU(ϕ₁, ϕ₂)
    T := \{s : ϕ₂ ∈ label(s)\}
    for all s ∈ T do
        label(s) := label(s) ∪ \{E[ϕ₁ U ϕ₂]\}
    end for
    while T ≠ {} do
        choose s ∈ T; T := T \ \{s\}
        for all t : R(t, s) do
            if E[ϕ₁ U ϕ₂] ∉ label(t) ∧ ϕ₁ ∈ label(t) then
                label(t) := label(t) ∪ \{E[ϕ₁ U ϕ₂]\}
                T := T ∪ \{t\}
            end if
        end for
    end while
end function
**Explicit CTL Model Checking: EG Operator**

```plaintext
function CHECKEG(\(\varphi_1\))
    \(S' := \{ s : \varphi_1 \in \text{label}(s) \}\)
    \(SCC = \{ C : C \) is non-trivial SCC of \(S' \}\)
    \(T := \bigcup_{C \in SCC} \{ s : s \in C \}\)
    for all \(s \in T\) do
        \(\text{label}(s) := \text{label}(s) \cup \{ \text{EG} \varphi_1 \}\)
    end for
    while \(T \neq \{\}\\) do
        choose \(s \in T\); \(T := T \setminus \{s\}\)
        for all \(t : t \in S' \land R(t, s)\) do
            if \(\text{EG} \varphi_1 \not\in \text{label}(t)\) then
                \(\text{label}(t) := \text{label}(t) \cup \{ \text{EG} \varphi_1 \}\)
                \(T := T \cup \{t\}\)
            end if
        end for
    end while
end function
```
Computing states satisfying particular sub-formulae:

- CheckEU: $O(|S| + |R|)$
- CheckEG: $O(|S| + |R|)$
  - Finding strongly connected components using Tarjan algorithm: $O(|S'| + |R'|)$
- EX: $O(|S| + |R|)$
- negation and conjunction: $O(|S|)$
- $\varphi$ contains at most $|\varphi|$ different sub-formulae

Total time complexity: $O(|\varphi| \times (|S| + |R|))$
DIFFERENCE BETWEEN CTL AND LTL

CTL and LTL are incomparable

- there are properties of one logic not expressible in the other one
- difference stems from their different semantics – while CTL captures sub-trees of computational tree, LTL considers each path in isolation
- both are useful, each in different settings
Theorem: There is no LTL formula equivalent to CTL formula $\text{AG} (\text{EF} p)$. 

Proof:
1. For contradiction assume there exists LTL formula $\phi$ equivalent to $\text{AG} (\text{EF} p)$.
2. State $s_0$ of KS(1) satisfies $\text{AG} (\text{EF} p)$. Therefore, $s_0$ satisfies $\phi$.
3. Since $\phi$ is satisfied in $s_0$, path looping in $s_0$ also satisfies it.
4. Therefore, state $s_0$ of KS(2) also satisfies $\phi$.
5. Since $\text{AG} (\text{EF} p)$ and $\phi$ are equivalent, state $s_0$ of KS(2) also satisfies $\text{AG} (\text{EF} p)$, which is contradiction.
Theorem: There is no LTL formula equivalent to CTL formula $AG(\text{EF } p)$.

Proof:
1. For contradiction assume there exists LTL formula $\varphi$ equivalent to $AG(\text{EF } p)$.
2. State $s_0$ of KS(1) satisfies $AG(\text{EF } p)$. Therefore, $s_0$ satisfies $\varphi$.
3. Since $\varphi$ is satisfied in $s_0$, path looping in $s_0$ also satisfies it.
4. Therefore, state $s_0$ of KS(2) also satisfies $\varphi$.
5. Since $AG(\text{EF } p)$ and $\varphi$ are equivalent, state $s_0$ of KS(2) also satisfies $AG(\text{EF } p)$, which is contradiction.
Theorem: There is no CTL formula equivalent to LTL formula \( F (G p) \).

- in particular it is not equivalent to \( AF (AG p) \)
**Theorem:** There is no CTL formula equivalent to LTL formula $F (G p)$.
- In particular, it is not equivalent to $AF (AG p)$

**Proof:**
- Consider Kripke structure below whose state $s_0$ satisfies $F (G p)$, but does not satisfy $AF (AG p)$.
COMPUTATIONAL-TREE PERSPECTIVE

\[ M, s_0 \models F(Gp) \]

\[ M, s_0 \not\models AF(AGp) \]
Consider producer and consumer communicating over reliable network:

Is this CTL formula satisfied in "OK" state?

\[ \text{AG} (\text{Sent} = \Rightarrow \text{AF} (\text{OK})) \]

No! Paths infinitely looping in "waiting" states violate it.
FAIRNESS CONSTRAINTS

Consider producer and consumer communicating over reliable network:

Is this CTL formula satisfied in “OK” state?

\[ \text{AG} (\text{Sent} \implies \text{AF} (\text{OK})) \]
Consider producer and consumer communicating over reliable network:

Is this CTL formula satisfied in “OK” state?

\[ AG (Sent \implies AF (OK)) \]

\textbf{No!} Paths infinitely looping in “waiting” states violate it.
We can define *fairness constraint* to avoid this types of failures:

![Diagram showing states and transitions]

- **OK**
- **Idle**
- **Sent**
- **Receive-data**
- **Send-data**
- **Produce-data**
- **Wait**
- **Process-data**
- **Send-ack**
- **Recv-ack**

When model checking, only fair paths are considered, those containing infinitely many fair states.
We can define *fairness constraint* to avoid this types of failures:

Introducing new AP *Idle*

Specifying fairness constraint as $\neg$*Idle*

When model checking, only *fair paths* considered – those containing infinitely many fair states.
Fair paths: introducing new atomic proposition *fair*:
- *fair* is true in state $s \leftrightarrow$ there exists fair path starting in $s$
- $M, s \models_F EG \text{ true}$
- Determining fair paths and deciding upon $EG p$ formulae

- $M, s \models_F p \leftrightarrow M, s \models p \land \text{fair}$
- $M, s \models_F EX \varphi \leftrightarrow M, s \models EX (\varphi \land \text{fair})$
- $M, s \models_F E[\varphi U \psi] \leftrightarrow M, s \models E[\varphi U (\psi \land \text{fair})]$
Complexity of CTL and LTL explicit model checking differs a bit:

- CTL: $O(|M| \ast |\varphi|)$
- LTL: $O(|M| \ast 2|\varphi|)$

Both linear in size of model, LTL exponential in size of formula

- practically negligible difference as formula is usually much smaller than model