4. Computation Tree Logic (3)

Huixing Fang

School of Information Engineering
Yangzhou University

Outline

1. Reduced OBDD
2. Implementation of ROBDD-Based Algorithms
3. CTL*

1 Reduced OBDD

Definition 1 (Reduced OBDD)

Let $\mathfrak{B}$ be a $\wp$-OBDD. $\mathfrak{B}$ is called **reduced** if for every pair (v, w) of nodes in $\mathfrak{B}$:

$$v \neq w \implies f_v \neq f_w.$$

Let $\wp$-ROBDD denote a reduced $\wp$-OBDD.

- In reduced $\wp$-OBDDs any $\wp$-consistent cofactor is represented by exactly one node
- This is the key property to prove that reduced OBDDs yield a universal and canonical data structure for switching functions

1 Reduced OBDD

Theorem 2 (Universality and Canonicity of ROBDDs)

Let Var be a finite set of Boolean variables and $\wp$ a variable ordering for Var. Then:

1. *For each switching function f for Var there exists a $\wp$-ROBDD $\mathfrak{B}$ with $f_{\mathfrak{B}} = f$*
2. *Given two $\wp$-ROBDDs $\mathfrak{B}$ and $\mathfrak{C}$ with $f_{\mathfrak{B}} = f_{\mathfrak{C}}$, then $\mathfrak{B}$ and $\mathfrak{C}$ are isomorphic, i.e., agree up to renaming of the nodes*

Corollary 3 (Minimality of Reduced OBDDs)

Let $\mathfrak{B}$ be a $\wp$-OBDD for f. Then, $\mathfrak{B}$ is reduced if and only if $size(\mathfrak{B}) \leq size(\mathfrak{C})$ for each $\wp$-OBDD $\mathfrak{C}$ for f.

1.1 Reduction Rules

When the variable ordering $\wp$ for *Var* is fixed, then **reduced** $\wp$-OBDDs provide **unique** representations of switching functions for *Var*

1. **Elimination rule:** If v is an **inner node** of $\mathfrak{B}$ with

$$succ_0(v) = succ_1(v) = w,$$

   then remove v and redirect all incoming edges $u \rightsquigarrow v$ to w

When the variable ordering $\wp$ for *Var* is fixed, then **reduced** $\wp$-OBDDs provide **unique** representations of switching functions for *Var*

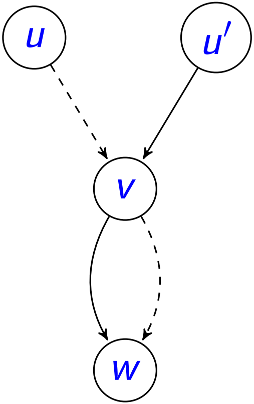
1. **Elimination rule:** If v is an **inner node** of $\mathfrak{B}$ with

$$succ_0(v) = succ_1(v) = w,$$

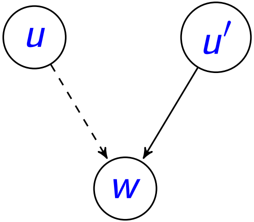
   then remove v and redirect all incoming edges $u \rightsquigarrow v$ to w
2. **Isomorphism rule:** If v, w are nodes in $\mathfrak{B}$ with $v \neq w$ and
   - either v, w are drains (i.e., terminal nodes) with $val(v) = val(w)$ or
   - v, w are inner nodes with

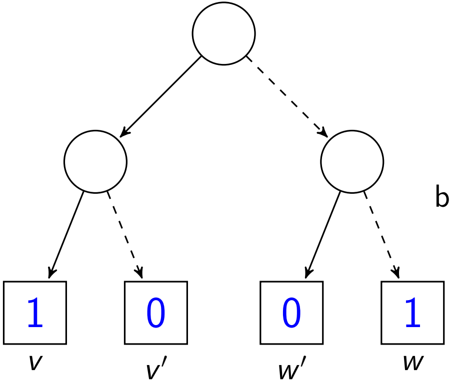
$$\langle var(v), succ_1(v), succ_0(v) \rangle = \langle var(w), succ_1(w), succ_0(w) \rangle$$

   then remove node v and redirect all incoming edges $u \rightsquigarrow v$ to node w

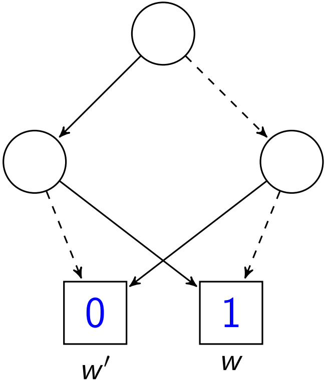


becomes

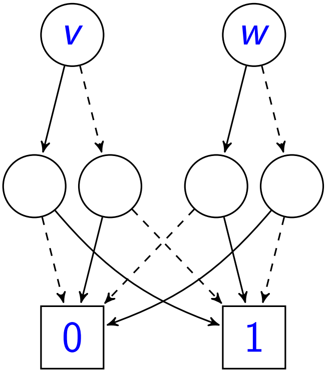




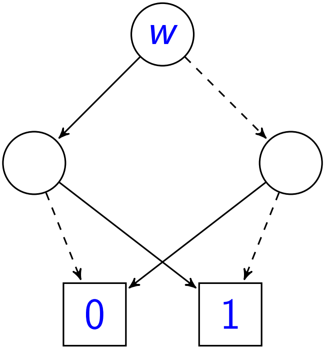
becomes



1.1 Reduction Rules



becomes



1.1 Reduction Rules

Both reduction rules are sound in the sense that they do not affect the semantics.

1. For the elimination rule applied to a z-node v with $w = succ_0(v) = succ_1(v)$, we have

$$f_v = (\neg z \wedge f_{succ_0(v)}) \wedge (z \wedge f_{succ_1(v)}) = (\neg z \wedge f_w) \wedge (z \wedge f_w) = f_w$$

2. If the isomorphism rule is applied to z-nodes v and w then

$$\begin{aligned} f_v &= (\neg z \wedge f_{succ_0(v)}) \wedge (z \wedge f_{succ_1(v)}) \\ &= (\neg z \wedge f_{succ_0(w)}) \wedge (z \wedge f_{succ_1(w)}) = f_w \end{aligned}$$

Theorem 4 (Completeness of Reduction Rules)

$\wp$-OBDD $\mathfrak{B}$ is reduced if and only if no reduction rule is applicable to $\mathfrak{B}$.

1 Reduced OBDD
- Reduction Rules
- Variable Ordering Problem

2 Implementation of ROBDD-Based Algorithms

3 CTL*

Let $m \geq 1$ and

$$f_m = (z_1 \wedge y_1) \vee (z_2 \wedge y_2) \vee \ldots \vee (z_m \wedge y_m).$$

1. For the variable ordering

$$\wp = (z_m, y_m, z_{m-1}, y_{m-1}, \ldots, z_1, y_1),$$

   the $\wp$-ROBDD for f_m has $2m + 2$ nodes,
2. while $\Omega(2^m)$ nodes are required for the ordering

$$\wp' = (z_1, z_2, \ldots, z_m, y_1, \ldots, y_m).$$

1.2 Variable Ordering Problem

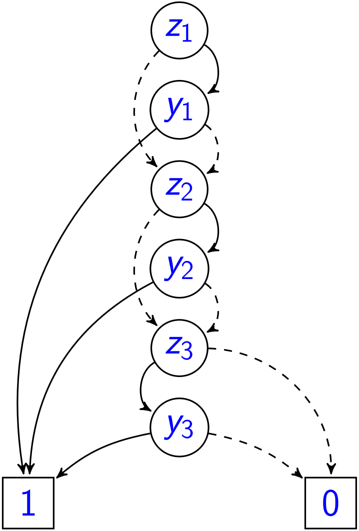


Figure: ROBDD for $f_3 = (z_1 \wedge y_1) \vee (z_2 \wedge y_2) \vee (z_3 \wedge y_3)$

1.2 Variable Ordering Problem

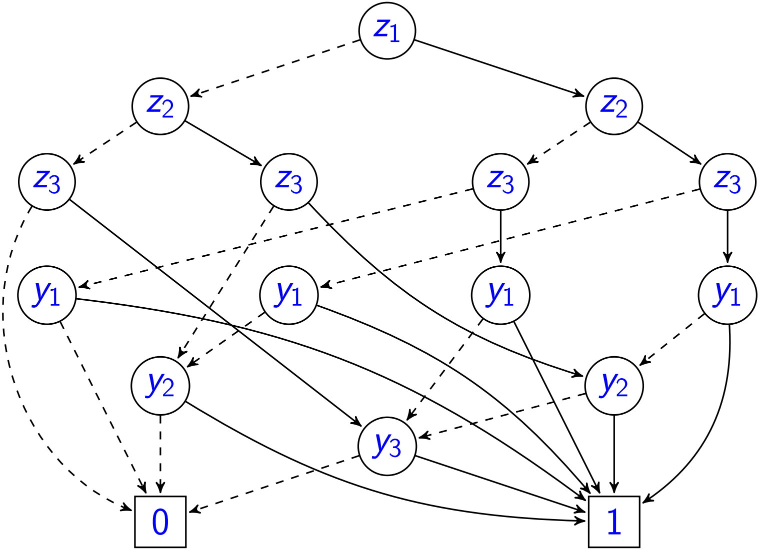


Figure: ROBDD for $f_3 = (z_1 \wedge y_1) \vee (z_2 \wedge y_2) \vee (z_3 \wedge y_3)$

1.2 Variable Ordering Problem

1. The size of ROBDDs is dependent on the variable ordering
2. To check whether a variable ordering is optimal is NP-hard
3. Many switching functions with large ROBDDs

Sifting algorithm

Dynamic variable ordering using variable swapping:

1. Select a variable x_i in OBDD
2. Successively swap x_i to determine $size(\mathfrak{B})$ at any position for x_i
3. Shift x_i to position for which $size(\mathfrak{B})$ is minimal
4. Go back to the first step until no improvement is made

1.2 Variable Ordering Problem

Symmetric Functions

$f \in Eval(z_1, ..., z_m 0)$ is **symmetric** if and only of

$$f([z_1 = b_1, ..., z_m = b_m]) = f([z_1 = b_{i_1}, ..., z_m = b_{i_m}])$$

for **each** permutation $(i_1, ..., i_m)$ of (1,...,m). Examples of symmetric functions:

1. $z_1 \vee z_2 \vee ... \vee z_m$
2. $z_1 \wedge z_2 \wedge ... \wedge z_m$
3. parity function, $z_1 \bigoplus z_2 \bigoplus ... \bigoplus z_m$, returns $\#(v \mapsto 1) = odd$
4. majority function, returns $\#(v \mapsto 1) > \#(v \mapsto 0)$

.

Lemma 5 (ROBDD-Size for Symmetric Functions)

If f is a symmetric function with m ***essential*** *variables, then for* ***each*** *variable ordering $\wp$ the $\wp$-ROBDD has size $O(m^2)$.*

1. Given a symmetric function f for m variables and $\wp = (z_1, \ldots, z_m)$,
2. then the $\wp$-consistent cofactors $f|z_1 = b_1, \ldots, z_i = b_i$ and $f|z_1 = c_1, \ldots, z_i = c_i$ agree for all bit tuples $(b_1, \ldots, b_i)$ and $(c_1, \ldots, c_i)$ that contain the same number of 1's.
3. Thus, there are at most $i + 1$ different $\wp$-consistent cofactors of f that arise by assigning values to the first i variables
4. Hence, the total number of $\wp$-consistent cofactors is bounded above by $\sum_{i=0}^{m}(i+1) = O(m^2)$ □

1.2 Variable Ordering Problem

ROBDDs vs. CNF/DNF

1. Given a CNF representation for f, the task to generate a CNF for $\neg f$ is expensive, CNF for $\neg f$ may have at least exponentially many clauses
2. However, negation is trivial as we may simply swap the values of the drains
3. For any variable ordering $\wp$, the $\wp$-ROBDDs for f and $\neg f$ have the same size
4. Regard the satisfiability problem, is NP-complete for CNF, but trivial for ROBDDs, since $f \neq 0$ if and only if the $\wp$-ROBDD for f does not contain a 0-drain
5. "linear time" for a ROBDD-based algorithm means linear in the size of the input ROBDDs

Outline

2 Implementation of ROBDD-Based Algorithms

The efficiency of ROBDD-based algorithms crucially relies on implementation techniques

1. Use a **single** reduced decision **graph** with **one global** variable ordering $\wp$ to represent several switching functions
2. All computations on these decision graphs are interleaved with the reduction rules to guarantee **redundance-freedom** at any time
3. The comparison of two represented **functions** simply requires checking equality of the **nodes** for them

1. Reduced OBDD

2. Implementation of ROBDD-Based Algorithms
   - Shared OBDD
   - Synthesizing shared ROBDDs

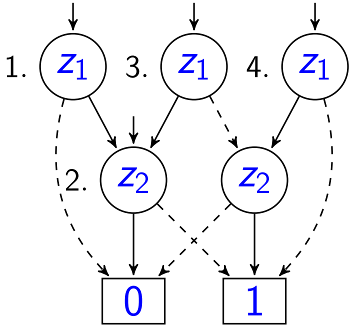
3. CTL*

2.1 Shared OBDD

Combine several OBDDs with same variable ordering such that common $\wp$-consistent co-factors are shared

Definition 6 (Shared OBDD)

- *Var*: a finite set of Boolean variables
- $\wp$ a variable ordering for Var
- **Shared** $\wp$-OBDD $\bar{\mathfrak{B}} = (V, V_I, V_T, succ_0, succ_1, var, val, \bar{v}_0)$
- $\bar{v}_0 = (v_0^1, \ldots, v_0^k)$ is a tuple of roots



$1.z_1 \wedge \neg z_2$, $2.\neg z_2$, $3.z_1 \oplus z_2$ and $4.\neg z_1 \vee z_2$

$\wp$-SOBDD is short for Shared $\wp$-OBDD

2.1 Shared OBDD

Symbolic CTL model-checking Φ:

1. the shared OBDD $\bar{\mathfrak{B}}$ with root nodes for Δ and
2. the f_a's, extended by new root nodes representing the characteristic functions of the satisfaction sets $Sat(\Psi)$, $\Psi \in Sub(\Phi)$
3. $\wp = (x_1, x'_1, ..., x_n, x'_n)$

2.2 Synthesizing shared ROBDDs

To support dynamic changes of the set of switching functions to be represented, two tables are required

1. **unique table:** contains the relevant information about the nodes and serves to keep the diagram reduced
2. **computed table:** for efficiency reasons

2.2 Synthesizing shared ROBDDs

The unique table

For each **inner node** v, the entries of the unique table are triples of the form

$$info(v) = \langle var(v), succ_1(v), succ_0(v) \rangle$$

1. contain the relevant information which is necessary for the applicability of the **isomorphism rule**
2. accessing via $op \triangleq find\_or\_add(z, v_1, v_0)$ with $v_1 \neq v_0$, and
3. return v **if there exists** a node $v = \langle z, v_1, v_0 \rangle$ in the ROBDD, or
4. create a **new** z-node v with $succ_0(v) = v_0$ and $succ_1(v) = v_1$
5. this op can be implemented using **hash functions**

ITE operator

ITE("if-then-else") operator for switching functions g, f_1, and f_2:

$$ITE(g, f_1, f_2) = (g \wedge f_1) \vee (\neg g \wedge f_2) \ .$$

1. ITE fits very well with the unique table by their info-triples

$$f_v = ITE(z, f_{succ_1(v)}, f_{succ_0 v}) \ .$$

2. $ITE(0, f_1, f_2) = f_2$, $ITE(1, f_1, f_2) = f_1$
3. $\neg f = ITE(f, 0, 1)$
4. $f_1 \vee f_2 = ITE(f_1, 1, f_2)$
5. $f_1 \wedge f_2 = ITE(f_1, f_2, 0)$
6. $f_1 \oplus f_2 = ITE(f_1, \neg f_2, f_2) = ITE(f_1, ITE(f_2, 0, 1), f_2)$
7. $f_1 \rightarrow f_2 = ITE(f_1, f_2, 1)$

Lemma 7 (Cofactors of ITE($\cdot$))

If g, f_1, f_2 are switching functions for Var, $z \in Var$ and $b \in \{0, 1\}$, then

$$ITE(g, f_1, f_2)|_{z=b} = ITE(g|_{z=b}, f_1|_{z=b}, f_2|_{z=b}) \ .$$

Proof: For any $(a, \bar{\mathfrak{c}}) = [z = a, \bar{y} = \bar{\mathfrak{c}}] \in Eval(Var)$

$$\begin{aligned}
& ITE(g, f_1, f_2)|_{z=b}(a, \bar{\mathfrak{c}}) \\
= \ & ITE(g, f_1, f_2)(b, \bar{\mathfrak{c}}) \\
= \ & (g(b, \bar{\mathfrak{c}}) \wedge f_1(b, \bar{\mathfrak{c}})) \vee (\neg g(b, \bar{\mathfrak{c}}) \wedge f_2(b, \bar{\mathfrak{c}})) \\
= \ & (g|_{z=b}(a, \bar{\mathfrak{c}}) \wedge f_1|_{z=b}(a, \bar{\mathfrak{c}})) \vee (\neg g|_{z=b}(a, \bar{\mathfrak{c}}) \wedge f_2|_{z=b}(a, \bar{\mathfrak{c}})) \\
= \ & ITE(g|_{z=b}, f_1|_{z=b}, f_2|_{z=b})(a, \bar{\mathfrak{c}}) \ .
\end{aligned}$$

$\square$

A node in a $\wp$-SOBDD for representing $ITE(g, f_1, f_2)$ is a node w such that $info(w) = \langle z, w_1, w_0 \rangle$ where

1. z is the minimal essential variable of $ITE(g, f_1, f_2)$ according to $<_\wp$
2. w_1, w_0 are SOBDD nodes with:

$$f_{w_1} = ITE(g|_{z=1}, f_1|_{z=1}, f_2|_{z=1})$$
$$f_{w_0} = ITE(g|_{z=0}, f_1|_{z=0}, f_2|_{z=0})$$

This suggests a recursive algorithm

1. determine $z = min\{var(u), var(v_1), var(v_2)\}$
2. recursively computes the nodes for ITE applied to the cofactors of $g = f_u, f_1 = f_{v_1}, f_2 = f_{v_2}$ for z

2.2 Synthesizing shared ROBDDs

Algorithm 1: $ITE(u, v_1, v_2)$(first version)

```
if u is terminal then
    if var(u) = 1 then
        w := v1;
    else
        w := v2;
    end
else
    z := min{var(u), var(v1), var(v2)};
    w1 := ITE(u|z=1, v1|z=1, v2|z=1);
    w0 := ITE(u|z=0, v1|z=0, v2|z=0);
    if w0 = w1 then
        w := w1;
    else
        w := find_or_add(z, w1, w0);
    end
end
return w;
```

Lemma 8 (ROBDD size of $ITE(g, f_1, f_2)$)

The size of the $\wp$-ROBDD for $ITE(g, f_1, f_2)$ is bounded by $N_g \cdot N_{f_1} \cdot N_{f_2}$ where N_f denotes the size of the $\wp$-ROBDD for f.

Proof: Let $\wp = (z_1, \ldots, z_m)$ and $\mathfrak{B}_f$ denote the $\wp$-ROBDD for f where the nodes are the $\wp$-consistent cofactors of f. The node set of $\mathfrak{B}_f$

$$V_f = \{f|_{z_1=b_1,\ldots,z_i=b_i} \mid 0 \leq i \leq m, b_1, \ldots, b_i \in \{0, 1\}\}$$

Note that several of the cofactors $f|z_1 = b_1, \ldots, z_i = b_i$ might agree.

Continue Proof:

- By Lemma 7, the node set $V_{ITE(g,f_1,f_2)}$ agrees with the set of switching functions

$$ITE(g|_{z_1=b_1,\ldots,z_i=b_i},\ f_1|_{z_1=b_1,\ldots,z_i=b_i},\ f_2|_{z_1=b_1,\ldots,z_i=b_i})$$

  where $0 \leq i \leq m$, $b_1, \ldots, b_i \in \{0, 1\}$.
- We can construct the surjective function

$$\iota : V_g \times V_{f_1} \times V_{f_2} \rightarrow V_{ITE(g,f_1,f_2)}.$$

  Hence,

$$\begin{aligned} N_{ITE(g,f_1,f_2)} &= |V_{ITE(g,f_1,f_2)}| \\ &\leq |V_g \times V_{f_1} \times V_{f_2}| = N_g \cdot N_{f_1} \cdot N_{f_2} \quad \square \end{aligned}$$

2.2 Synthesizing shared ROBDDs

Algorithm 2: $ITE(u, v_1, v_2)$(Use Computed Table)

if *there is an entry for* (u, v_1, v_2, w) *in the* ***computed table*** **then**
    return w;
else
    if u *is terminal* **then**
        if $var(u) = 1$ **then** $w := v_1$ **else** $w := v_2$;
    else
        $z := \min\{var(u), var(v_1), var(v_2)\}$;
        $w_1 := ITE(u|_{z=1}, v_1|_{z=1}, v_2|_{z=1})$;
        $w_0 := ITE(u|_{z=0}, v_1|_{z=0}, v_2|_{z=0})$;
        if $w_0 = w_1$ **then** $w := w_1$ **else** $w := find\_or\_add(z, w_1, w_0)$;
        insert (u, v_1, v_2, w) in the **computed table**;
    end
    return w;
end

Outline

1. **CTL*** is an extension of CTL as it allows path quantifiers $\exists$ and $\forall$ to be **arbitrarily nested with linear temporal operators** such as $\bigcirc$ and $\mathbf{U}$
   - $\forall\bigcirc\bigcirc a$ is a legal CTL* formula
   - but $\forall\bigcirc\bigcirc a$ is not in CTL
2. In contrast, in **CTL** each linear temporal operator must be **immediately preceded** by a path quantifier

CTL* **state formulae** Φ over AP, briefly called CTL* formulae,

$$\Phi ::= \mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi$$

where $a \in AP$ and φ is a **path formula**:

$$\varphi ::= \Phi \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \mathbf{U} \varphi_2$$

where Φ is a **state formula**, and φ, φ_1, and φ_2 are **path formulae**

$$\boxed{\Diamond\varphi = \mathit{true}\,\mathbf{U}\,\varphi, \quad \Box\varphi = \neg\Diamond\neg\varphi, \quad \forall\varphi = \neg\exists\neg\varphi}$$

The satisfaction relation $\models$ is defined for state formulae by

$$
\begin{array}{lll}
s \models a & \text{iff} & a \in L(s), \\
s \models \neg\Phi & \text{iff} & \text{not } s \models \Phi, \\
s \models \Phi \wedge \Psi & \text{iff} & (s \models \Phi) \text{ and } (s \models \Psi) \\
s \models \exists\varphi & \text{iff} & \pi \models \varphi \text{ for some } \pi \in Paths(s).
\end{array}
$$

For path $\pi = s_0 s_1 s_2 \ldots$, the satisfaction relation $\models$ for path formulae is defined by:

$$
\begin{array}{lll}
\pi \models \Phi & \text{iff} & s_0 \models \Phi, \\
\pi \models \varphi_1 \wedge \varphi_2 & \text{iff} & (\pi \models \varphi_1) \text{ and } (\pi \models \varphi_2), \\
\pi \models \neg\varphi & \text{iff} & \pi \not\models \varphi, \\
\pi \models \bigcirc\varphi & \text{iff} & \pi[1..] \models \varphi, \\
\pi \models \varphi_1 \mathbf{U} \varphi_2 & \text{iff} & \exists j \geq 0.\ (\pi[j..] \models \varphi_2 \wedge \\
 & & \quad (\forall 0 \leq k < j.\ \pi[k..] \models \varphi_1))
\end{array}
$$

- $\pi[i..]$ denotes the suffix of π from index $i \geq 0$ on

For CTL*-state formula Φ, the satisfaction set $Sat(\Phi)$ is defined by

$$Sat(\Phi) = \{s \in S \mid s \models \Phi\} .$$

The transition system TS satisfies CTL* formula Φ if and only if Φ holds in **all initial states** of TS:

$$TS \models \Phi \quad \text{if and only if} \quad \forall s_0 \in I.\ s_0 \models \Phi .$$

3.4 Embedding of LTL in CTL*

For each LTL formula φ over AP and for each $s \in S$:

$$\underbrace{s \models \varphi}_{\text{LTL semantics}} \quad \text{iff} \quad \underbrace{s \models \forall\varphi}_{\text{CTL* semantics}}$$

$$\underbrace{TS \models \varphi}_{\text{LTL semantics}} \quad \text{iff} \quad \underbrace{TS \models \forall\varphi}_{\text{CTL* semantics}}$$

It is justified to understand LTL as a sublogic of CTL*

Theorem 9 (CTL* is More Expressive Than LTL and CTL)

For the CTL formula over* $AP = \{a, b\}$,

$$\Phi = (\forall\Diamond\Box a) \vee (\forall\Box\exists\Diamond b) \ ,$$

there does not exist any equivalent LTL or CTL formula

Proof:

1. $\forall\Box\exists\Diamond b$ is a CTL formula but not in LTL
2. $\Diamond\Box a$ is an LTL formula but not in CTL

$\Box$

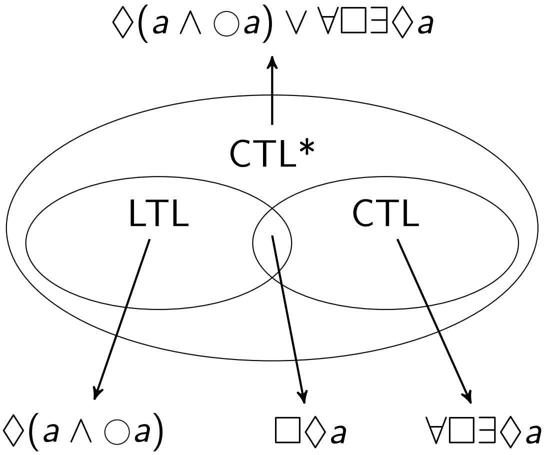


Figure: Relationship between LTL, CTL and CTL* duality laws for path quantifiers

$$\neg\forall\varphi \equiv \exists\neg\varphi$$
$$\neg\exists\varphi \equiv \forall\neg\varphi$$

distributive laws

$$\forall(\varphi_1 \wedge \varphi_2) \equiv \forall\varphi_1 \wedge \forall\varphi_2$$
$$\exists(\varphi_1 \vee \varphi_2) \equiv \exists\varphi_1 \vee \exists\varphi_2$$

quantifier absorption laws

$$\forall\Box\Diamond\varphi \equiv \forall\Box\forall\Diamond\varphi$$
$$\exists\Diamond\Box\varphi \equiv \exists\Diamond\exists\Box\varphi$$

CTL$^+$ extends CTL by allowing Boolean operators in path formulae:

1. CTL$^+$ state formulae Φ

$$\Phi ::= \mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$

2. CTL$^+$ path formulae φ

$$\varphi ::= \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\Phi \mid \Phi_1 \mathbf{U} \Phi_2$$

3. CTL$^+$ is as expressive as CTL

For any CTL$^+$ state formula Φ^+ there exists an equivalent CTL formula Φ. For example:

$$\underbrace{\exists(a\,\mathbf{W}\,b)}_{\text{CTL formula}} \equiv \underbrace{\exists((a\,\mathbf{U}\,b) \vee \Box a)}_{\text{CTL}^+\text{ formula}}$$

$$\underbrace{\exists(\Diamond a \wedge \Diamond b)}_{\text{CTL}^+\text{ formula}} \equiv \underbrace{\exists\Diamond(a \wedge \exists\Diamond b) \wedge \exists\Diamond(b \wedge \exists a)}_{\text{CTL formula}}$$

The transformation relies on equivalence laws such as

$$\begin{aligned}
\exists(\neg\bigcirc\Phi) &\equiv \exists\bigcirc\neg\Phi \\
\exists(\neg(\Phi_1\,\mathbf{U}\,\Phi_2)) &\equiv \exists((\Phi_1 \wedge \neg\Phi_2)\,\mathbf{U}\,(\neg\Phi_1 \wedge \neg\Phi_2)) \vee \exists\Box\neg\Phi_2 \\
\exists(\bigcirc\Phi_1 \wedge \bigcirc\Phi_2) &\equiv \exists\bigcirc(\Phi_1 \wedge \Phi_2) \\
&\vdots
\end{aligned}$$

1. CTL can be expanded by means of a Boolean operator for path formulae without changing the expressiveness
2. CTL$^+$ formulae can be much shorter than the shortest equivalent CTL formulae

The model-checking algorithm for CTL* is based on a bottom-up traversal of the syntax tree of Φ

Definition 10 (Maximal Proper State Subformula)

State formula Ψ is a maximal proper state subformula of Φ whenever

1. Ψ is a subformula of that differs from Φ
2. and Ψ is not contained in any other proper state subformula of Φ

Recursive computation of satisfaction sets:

$$
\begin{aligned}
Sat(true) &= S \\
Sat(a) &= \{s \in S \mid a \in L(s)\} \\
Sat(\Phi_1 \wedge \Phi_2) &= Sat(\Phi_1) \cap Sat(\Phi_2) \\
Sat(\neg\Phi) &= S \setminus Sat(\Phi) \\
Sat(\forall\varphi) &= Sat_{LTL}(\varphi) \\
Sat(\exists\varphi) &= S \setminus Sat_{LTL}(\neg\varphi)
\end{aligned}
$$

Φ_1 and Φ_2 are maximal proper state subformulae of Φ:

$$\Phi = \underbrace{\exists\Diamond\Box a}_{\Phi_1} \wedge \exists\Box(\bigcirc b \wedge \Diamond \underbrace{\neg\exists(a\,\mathbf{U}\,b)}_{\Phi_2})$$

1. calculate recursively the satisfaction sets $Sat(\Phi_i)$
2. replace Φ_i with the atomic proposition a_i, $i = 1, 2$

$$\Phi \rightsquigarrow a_1 \wedge \exists \underbrace{\Box(\bigcirc b \wedge \Diamond a_2)}_{\text{LTL formula } \varphi} = a_1 \wedge \exists\varphi$$

3. compute $Sat(\exists\varphi)$ via NBA $\mathcal{A}$ for φ and nested DFS in $TS \oplus \mathcal{A}$
4. return $Sat(\Phi) = Sat(a_1 \wedge \exists\varphi) = Sat(a_1) \cap Sat(\exists\varphi)$

3.7 CTL* Model Checking

Algorithm 3: CTL* model checking algorithm (basic idea)

Input: TS with initial states I, and CTL* formula Φ
Output: $I \subseteq Sat(\Phi)$

foreach $i \leq |\Phi|$ **do**
  foreach $\Psi \in Sub(\Phi)$ *with* $|\Psi| = i$ **do**
    switch Ψ **do**
      case *true* **do** $Sat(\Psi) := S$;
      case a **do** $Sat(\Psi) := \{s \in S \mid a \in L(s)\}$;
      case $a_1 \wedge a_2$ **do** $Sat(\Psi) := Sat(a_1) \cap Sat(a_2)$;
      case $\neg a$ **do** $Sat(\Psi) := S \setminus Sat(a)$;
      case $\exists\varphi$ **do** $Sat(\Psi) := S \setminus Sat_{LTL}(\neg\varphi)$;
    end
    $AP := AP \cup \{a_\Psi\}$;
    replace Ψ with a_Ψ;
    foreach $s \in Sat(\Psi)$ **do** $L(s) := L(s) \cup \{a_\Psi\}$;
  end
end
return $I \subseteq Sat(\Phi)$;

3.7 CTL* Model Checking

Evidently, the time complexity of the CTL* model-checking algorithm is dominated by the LTL model-checking phases

Theorem 11 (Time Complexity of CTL* Model Checking)

For transition system TS with N states and K transitions, and CTL formula Φ, the CTL* model-checking problem $TS \models \Phi$ can be determined in time $O((N + K) \cdot 2^{|\Phi|})$.*

- CTL* model checking can be solved by any LTL model-checking algorithm
- The CTL* model-checking problem is PSPACE-complete

Summary

1. CTL is a logic for formalizing properties over computation trees, i.e., the branching structure of the states
2. The expressivenesses of LTL and CTL are incomparable
3. The CTL model-checking problem can be solved by a recursive descent procedure over the parse tree of the state formula to be checked
4. The time complexity of the CTL model-checking algorithm is linear in the size of the transition system and the length of the formula
5. The CTL model-checking procedure can be realized symbolically by means of ordered binary decision diagrams
6. CTL* is more expressive than either CTL and LTL
7. The CTL* model-checking problem can be solved by an appropriate combination of the recursive descent procedure (as for CTL) and the LTL model-checking algorithm, PSPACE-complete