Satisfiability Checking of Temporal Logics for Verification of Pointer Systems

Yoshinori Tanabe
Japan Science and Technology Agency (CREST)
Research Center for Verification and Semantics, AIST

CVS/AIST Workshop
on Verification and Rewriting
October 21, 2004

Backgound

Background:

- State Explosion Problem, especially in software model checking.
- Abstraction is a key issue.
- Various existing tools.
  - Bandera (Java), SLAM (C), Blast (C), ...
- Few handle properties on structures built up by pointers.
- Hagiya-Takahashi proposed a method of abstraction using temporal formula.

Goal: A tool that supports abstraction to make model checking on pointer systems feasible.

Predicate Abstraction

Pointer System

Satisfiability Checking

Hybrid Temporal Logic

Predicate Abstraction

Summary:

- A method to build an abstract Kripke structure from a given (possibly infinite) concrete Kripke structure.
- It requires:
  - a set of predicates,
  - a procedure to calculate a precondition of a predicate, and
  - a decision procedure for satisfiability of predicates.
- The resulting abstraction is sound for ACTL* properties.

Predicate Abstraction (cont)

AP: the set of atomic propositions

Kripke structure: $\mathcal{C} = (S, R, L)$

- S: the set of states
- $R \subseteq S \times S$: the transition relation
- $L : \mathsf{AP} \to \mathscr{P}(S)$: the labelling function

$\mathcal{Q} := \{Q_i \mid i = 1, \ldots, n\}$: a set of predicates on S. i.e. for each $s \in S$, either Q_i is true or false at s.

Predicate Q is a precondition of predicate Q' if

$$(s, s') \in R,\ Q' \text{ is true at } s' \implies Q \text{ is true at } s.$$

A decision procedure DP for satisfiability checking of predicates should behave as follows:

- If Q is satisfiable, i.e. $\exists s \in S.\, Q$ is true at s, then DP should reply "yes",
- otherwise, DP should reply "yes" or "no".

Suppose that from predicate Q we can compute one of its precondition. Let pre(Q) be the result.

Abstract Trasition System

The abstract transition system $(A, \overline{R})$ is defined as follows:

- $A := \mathscr{P}(\mathcal{Q})$
- For $a, a' \in A$, $(a, a') \in \overline{R} \iff$ DP replies "yes" for $\bigwedge\{Q \mid Q \in a\} \wedge \bigwedge\{\neg Q \mid Q \in \mathcal{Q} \setminus a\} \wedge \bigwedge\{\mathsf{pre}(Q') \mid Q' \in a'\} \wedge \bigwedge\{\mathsf{pre}(\neg Q') \mid Q' \in \mathcal{Q} \setminus a'\}$

Mapping $\beta : S \to A$ can be defined naturally:

$$\beta(s) = \{Q \in \mathcal{Q} \mid Q \text{ is true at } s\}$$

φ_0: an ACTL* formula

AP$'$: the set of atomic propositions appeared in φ_0.

Q_p: predicate "$s \in L(p)$", for $p \in$ AP$'$.

Suppose for each $p \in$ AP$'$, $Q_p \in \mathcal{Q}$. Then $(A, \overline{R})$ is naturally extended to a Kripke structure $\mathcal{A} = (A, \overline{R}, \overline{L})$:

$$a \in \overline{L}(p) \iff p \in \text{AP}' \text{ and } Q_p \in a.$$

Theorem

$\mathcal{A}$ is a sound abstraction: for $s \in S$,

$$\mathcal{A}, \beta(s) \models \varphi_0 \implies \mathcal{C}, s \models \varphi_0,$$

hence

$$\mathcal{A} \models \varphi_0 \implies \mathcal{C} \models \varphi_0.$$

Predicate Abstraction

Pointer System

Satisfiability Checking

Hybrid Temporal Logic

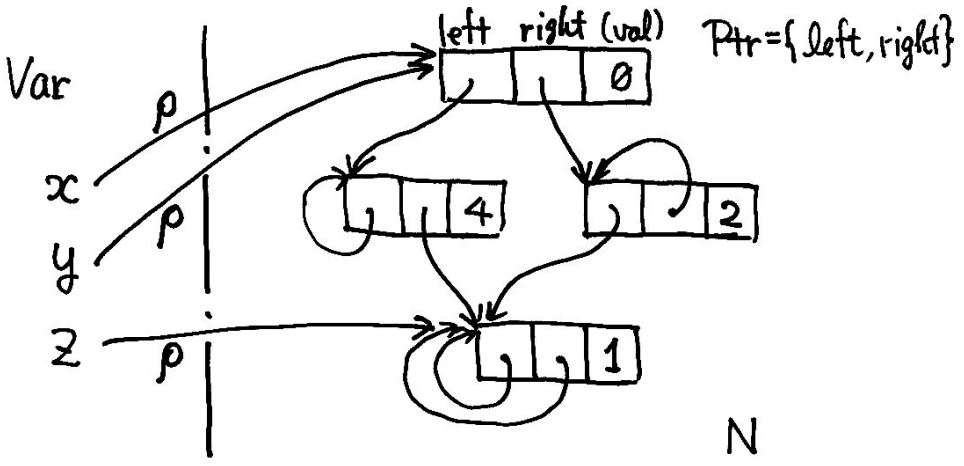
Pointer Structure

Var: the set of variables

Ptr: the set of (names of) pointers

pointer structure: tuple (N, p, v, ρ) s.t.

- N is the set of nodes.
- $p : \mathsf{Ptr} \to N \to N$
- $v : N \to \omega$
- $\rho : \mathsf{Var} \to N$



Program

statement: one of the following, possibly with a label

- `if (`*cond*`) goto` *label*
- x `:=` y
- x `:=` y`.`p
- x `:= new()`
- x`.val :=` n
- x`.`p `:=` y

where $x, y \in$ Var; $n \in \omega$; $p \in$ Ptr.

cond is x `==` y`,` x`.val ==` n`,` or their boolean combination.

program: finite sequence of statements

Transition system

PtrStr := the class of all pointer structures

PtrStr $\times \omega$ can be regarded as a transition system by a program P:

> $(S, i) \to (S', i') \overset{\text{def}}{\iff}$ "Executing the i-th statement of P on S, the resulting structure is S' and the next to be executed is the i'-th statement."

This is the target system on which we will define predicates, calculate preconditions, and find a decision procedure for satisfiability.

Two-way CTL

Mod: the set of modalities

For $a \in \mathrm{Mod}$, $\overline{a} \in \mathrm{Mod}$, $\overline{\overline{a}} = a$

two-way CTL formula φ:

$$\varphi = p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{E}_A\mathbf{X}\,\varphi \mid \mathbf{A}_A\mathbf{X}\,\varphi$$
$$\mid \mathbf{E}_A\mathbf{F}\,\varphi \mid \mathbf{A}_A\mathbf{F}\,\varphi \mid \mathbf{E}_A\mathbf{G}\,\varphi \mid \mathbf{A}_A\mathbf{G}\,\varphi$$

($p \in \mathrm{AP}$, $A \subseteq \mathrm{Mod}$, $0 < |A| < \omega$)

Form := the set of two-way CTL formulas.

Kripke structure $K = (S, R, L)$.

- $R : \mathrm{Mod} \to \mathscr{P}(S \times S)$
- $(s, s') \in R(a) \iff (s', s) \in R(\overline{a})$
- $L : \mathrm{AP} \to \mathscr{P}(S)$

Satisfaction Relation

- For formula φ and $s \in S$,
  - $s \models p \iff s \in L(p)$
  - $s \models \neg p \iff s \notin L(p)$
  - $s \models \varphi_1 \vee \varphi_2 \iff s \models \varphi_1$ or $s \models \varphi_2$
  - $s \models \varphi_1 \wedge \varphi_2 \iff s \models \varphi_1$ and $s \models \varphi_2$
  - $s \models \mathbf{E}_A \mathbf{X}\, \varphi \iff \exists s' \in S, \exists a \in A.\ (s, s') \in R_a,\ s' \models \varphi.$
  - $s \models \mathbf{A}_A \mathbf{X}\, \varphi \iff$
    $$\forall s' \in S, \forall a \in A.\ (s, s') \in R_a \implies s' \models \varphi.$$
  - $s \models \mathbf{E}_A \mathbf{F}\, \varphi \iff \exists \sigma\colon A\text{-path}.\ \sigma_0 = s, \exists i.\ \sigma_i \models \varphi$
  - $s \models \mathbf{A}_A \mathbf{F}\, \varphi \iff \forall \sigma\colon A\text{-path}.\ \sigma_0 = s \implies \exists i.\ \sigma_i \models \varphi$
  - $s \models \mathbf{E}_A \mathbf{G}\, \varphi \iff \exists \sigma\colon A\text{-path}.\ \sigma_0 = s, \forall i.\ \sigma_i \models \varphi$
  - $s \models \mathbf{A}_A \mathbf{G}\, \varphi \iff \forall \sigma\colon A\text{-path}.\ \sigma_0 = s \implies \forall i.\ \sigma_i \models \varphi$

Formula and Kripke Structure for Pointer Structure

AP $:=$ Var $\cup\, \omega$, Mod $:=$ Ptr $\cup\, \{\overline{a} \mid a \in \mathsf{Ptr}\}$

Formula Examples: ($x, y, z, \mathsf{nil} \in \mathsf{Var}$; $a \in \mathsf{Ptr}$)

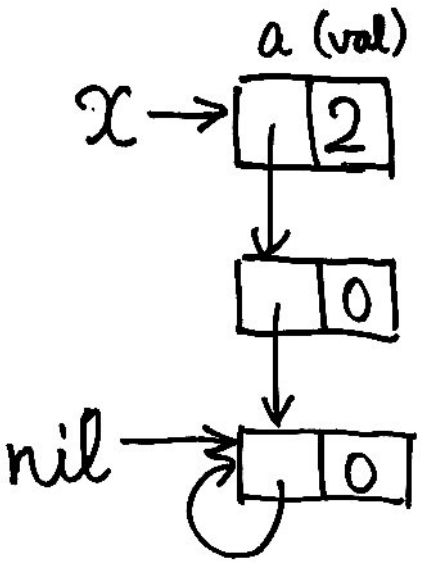
- $x \wedge 2 \wedge \mathbf{E}_a\mathbf{F}\ \mathsf{nil}$
- $x \wedge \mathbf{E}_a\mathbf{X}\,\mathbf{E}_a\mathbf{F}\ x$
- $z \wedge \mathbf{E}_{\overline{a}}\mathbf{X}\ x \wedge \mathbf{E}_{\overline{a}}\mathbf{X}\ y$

We can naturally define a Kripke structure $K(S) = (N, R, L)$ from a pointer structure $S = (N, p, v, \rho)$:

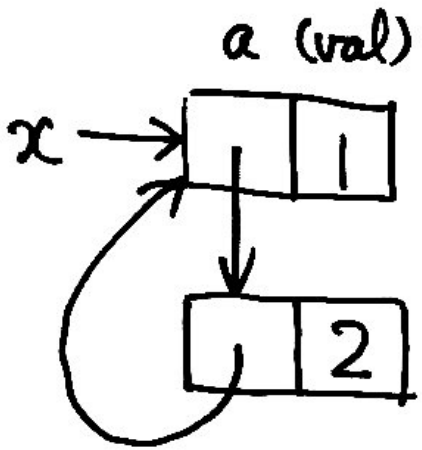
- $R(a) = \{(n, n') \mid p(a)(n) = n'\}$
- $R(\overline{a}) = \{(n', n) \mid p(a)(n) = n'\}$
- $L(x) = \{\rho(x)\}$ $\qquad (x \in \mathsf{Var})$
- $L(m) = \{n \in N \mid v(n) = m\}$ $\qquad (m \in \omega)$

Examples:

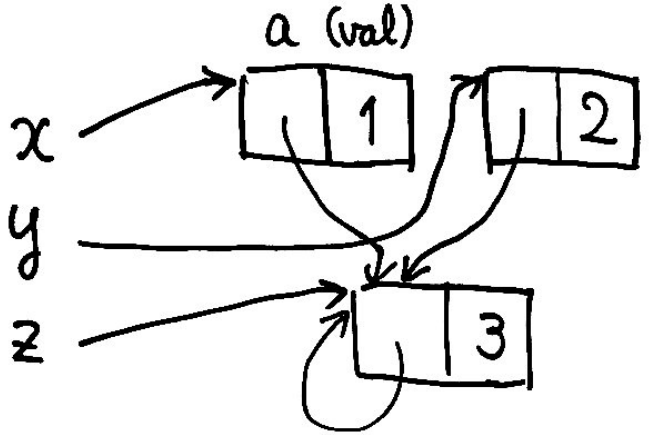
$x \wedge 2 \wedge \mathbf{E}_a\mathbf{F}$ nil



$x \wedge \mathbf{E}_a\mathbf{X}\,\mathbf{E}_a\mathbf{F}\,x$



$z \wedge \mathbf{E}_{\overline{a}}\mathbf{X}\,x \wedge \mathbf{E}_{\overline{a}}\mathbf{X}\,y$



Predicate

We define predicates for the abstraction:

1. $\exists\varphi$ is a predicate for a formula φ.
2. $\neg Q_1$ and $Q_1 \vee Q_2$ are predicates for predicates Q_1 and Q_2.

For a Kripke structure K and a predicate Q, we define $K \models^* Q$:

- $K \models^* \exists\varphi \iff$ There exists $n \in N$ s.t. $K, n \models \varphi$.
- $K \models^* \neg Q \iff K \not\models^* Q$.
- $K \models^* Q_1 \vee Q_2 \iff K \models^* Q_1$ or $K \models^* Q_2$.

For a pointer structure S and a predicate Q,

$$S \models^* Q \overset{\text{def}}{\iff} K(S) \models^* Q$$

Abbrev: $Q_1 \wedge Q_2$, $Q_1 \rightarrow Q_2$, $Q_1 \leftrightarrow Q_2$, $\forall\varphi(= \neg\exists\neg\varphi)$

Precondition

Theorem

(1) When P_i is "`if` c `goto` l," there is a procedure to construct a predicate Q from P_i such that

$$c \text{ holds on } S \iff S \models^{*} Q.$$

(2) When P_i is not an if-statement, there is a procedure to construct a predicate Q from a given predicate Q' such that for any $S, S' \in \mathsf{PtrStr}$ with $(S, i) \to_P (S', i+1)$:

$$S \models^{*} Q \iff S' \models^{*} Q'.$$

So far we have proved it only for (one-way) CTL.

Decision Procedure

We need a decision procedure DP such that for a given predicate Q,

- When Q is satisfied by a pointer structure, it returns "yes,"
- otherwise it returns "yes" or "no." (But we want it to return "no" as much as possible.)

A candidate for DP

Suppose we have a complete decision procedure DP' for satisfiability of a formula in two-way CTL.

For predicate $Q = \bigwedge_i \{\exists \varphi_i\} \wedge \bigwedge_j \{\forall \psi_j\}$,

- For φ_i; if its form is $\varphi_i = x_i \wedge \varphi_i'$ ($x_i \in \mathrm{Var}$), let $\varphi_i' := \mathbf{A}_A \mathbf{G}\,(x_i \rightarrow \varphi_i')$, otherwise $\varphi_i' :=$ true, where A is the set of all modalities appeared in Q.
  $\chi_1 := \bigwedge_i \varphi_i'$
- Let φ'' be any of φ_i that is not in the form above. If no such φ_i exists, $\varphi'' :=$ true.
- $\chi_2 := \bigwedge_j \mathbf{A}_A \mathbf{G}\, \psi_j$.

If predicate Q is satisfiable, then formula $\chi_1 \wedge \chi_2 \wedge \varphi''$ is satisfiable. (The converse does not hold.)

DP: "Apply DP' to formula $\chi_1 \wedge \chi_2 \wedge \varphi$."

Problems

1. If there are lots of $\exists\varphi$ that cannot be converted to $\mathbf{AG}$ formulas, precision might be low. (Important information might be lost.)
2. Even if Q is satisfiable, i.e. it is satisfied by a Kripke structure, it may not be satisfied by a pointer structure.

A Kripke structure $K(S) = (N, R, \lambda)$ obtained from a pointer structure has properties:

- For each $x \in \mathsf{Var}$, there is unique $n \in N$ such that $n \models x$.
- For $n \in N$ and $a \in \mathsf{Ptr}$, there is unique $m \in N$ such that $(n, m) \in R(a)$.

Predicate Abstraction

Pointer System

Satisfiability Checking

Hybrid Temporal Logic

Expansion, Closure, ...

- expansion
  - $\exp(\mathbf{E}_A\mathbf{F}\,\varphi) := \varphi \vee \mathbf{E}_A\mathbf{X}\,\mathbf{E}_A\mathbf{F}\,\varphi$
  - $\exp(\mathbf{A}_A\mathbf{F}\,\varphi) := \varphi \vee \mathbf{A}_A\mathbf{X}\,\mathbf{A}_A\mathbf{F}\,\varphi$
  - $\exp(\mathbf{E}_A\mathbf{G}\,\varphi) := \varphi \wedge \mathbf{E}_A\mathbf{X}\,\mathbf{E}_A\mathbf{F}\,\varphi$
  - $\exp(\mathbf{A}_A\mathbf{G}\,\varphi) := \varphi \wedge \mathbf{A}_A\mathbf{X}\,\mathbf{A}_A\mathbf{G}\,\varphi$
- $\varphi \equiv \exp(\varphi)$
- closure
  - $\varphi \in \mathsf{cl}(\varphi)$
  - $\mathsf{op}(\varphi_1, \ldots) \in \mathsf{cl}(\varphi) \implies \varphi_1 \in \mathsf{cl}(\varphi), \ldots$
    $(\mathsf{op} = \neg, \wedge, \vee, \mathbf{E}_A\mathbf{X}\,, \mathbf{A}_A\mathbf{X}\,, \mathbf{E}_A\mathbf{F}\,, \mathbf{A}_A\mathbf{F}\,, \mathbf{E}_A\mathbf{G}\,, \mathbf{A}_A\mathbf{G}\,)$
  - $\varphi_1 \in \mathsf{cl}(\varphi)$, $\exp(\varphi_1)$: defined $\implies \exp(\varphi_1) \in \mathsf{cl}(\varphi)$
- $\mathsf{Bas}(\varphi) := \{\psi \in \mathsf{cl}(\varphi) \mid \psi \in \mathsf{AP}$ or $\psi = \mathbf{E}_A\mathbf{X}\,\psi'$ or $\psi = \mathbf{A}_A\mathbf{X}\,\psi'\}$

Note: The truth value of $\psi \in \mathsf{cl}(\varphi)$ can be decided from those of $\mathsf{Bas}(\varphi)$.

E.g. $p \wedge \mathbf{E}_A\mathbf{F}\, q \equiv p \wedge (q \vee \mathbf{E}_A\mathbf{X}\, \mathbf{E}_A\mathbf{F}\, q)$

φ_0-type

φ_0: the formula the satisfiability of which we will judge

Type $:= \mathscr{P}(\mathrm{Bas}(\varphi_0))$: the set of φ_0-type.

For $t \in$ Type and $\varphi \in \mathrm{cl}(\varphi_0)$, we define $t \Vdash \varphi$:

- $t \Vdash \varphi \iff \varphi \in t$ (for $\varphi \in$ Bas)
- $t \Vdash \neg p \iff p \notin t$ (for $p \in$ AP)
- $t \Vdash \varphi \vee \psi \iff t \Vdash \varphi$ or $t \Vdash \psi$
- $t \Vdash \varphi \wedge \psi \iff t \Vdash \varphi$ and $t \Vdash \psi$
- $t \Vdash \varphi \iff t \Vdash \exp(\varphi)$ (if $\exp(\varphi)$ is defined.)

For $a \in$ Mod, we define $R(a) \subseteq$ Type $\times$ Type:
$(t, t') \in R(a) \iff$

- $\forall \mathbf{A}_A \mathbf{X}\, \varphi \in \mathrm{cl}(\varphi_0).\ (t \Vdash \mathbf{A}_A \mathbf{X}\, \varphi, a \in A) \implies t' \Vdash \varphi$
- $\forall \mathbf{A}_A \mathbf{X}\, \varphi \in \mathrm{cl}(\varphi_0).\ (t' \Vdash \mathbf{A}_A \mathbf{X}\, \varphi, \overline{a} \in A) \implies t \Vdash \varphi$
- $\forall \mathbf{A}_A \mathbf{F}\, \varphi \in \mathrm{cl}(\varphi_0).\ (a \in A, \overline{a} \in A, t \Vdash \mathbf{A}_A \mathbf{F}\, \varphi, t' \Vdash \mathbf{A}_A \mathbf{F}\, \varphi) \implies$
  $t \Vdash \varphi$ or $t' \Vdash \varphi$

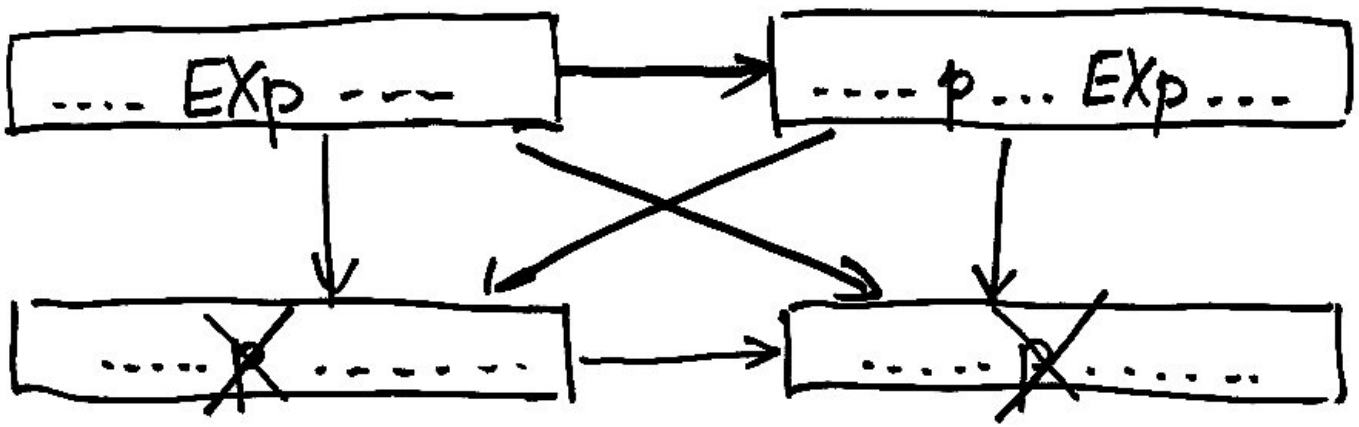
Algorithm

1. Start with $T := \text{Type}$.
2. Remove $t \in T$ from T if t is either EX-inconsistent, EF-inconsistent, or AF-inconsistent in T.
3. Repeat above as much as possible.
4. If there exists $t \in T$ such that $t \Vdash \varphi_0$, then φ_0 is satisfiable, otherwise it is not satisfiable.

EX-inconsistency

$t \in T$ is EX-insonsistent in T

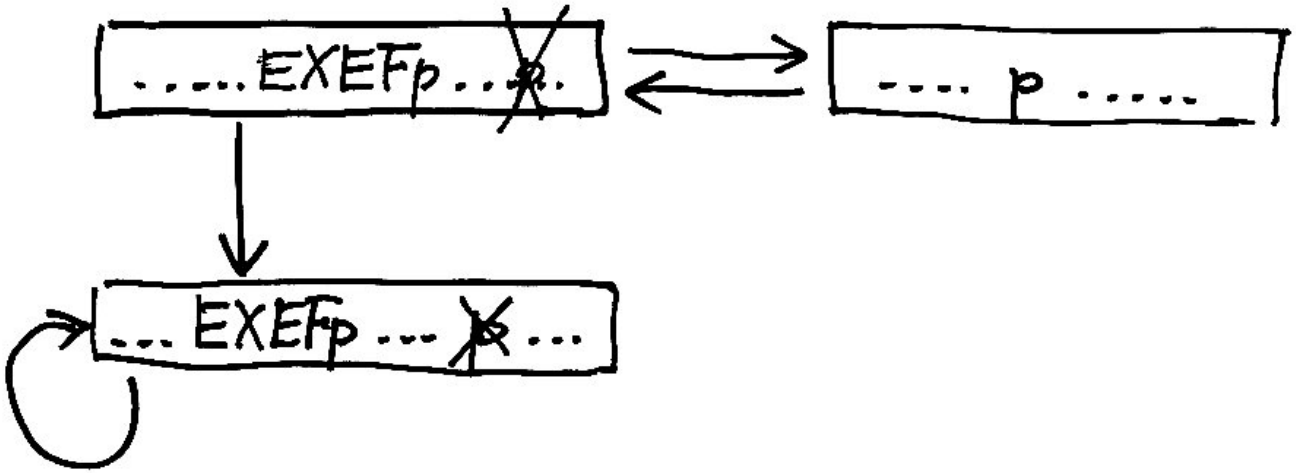
$\stackrel{\text{def}}{\iff} \exists \mathbf{E}_A \mathbf{X}\, \varphi \in t.\ \ \forall t' \in T. \forall a \in A.\ \ (t,t') \in R(a) \implies t' \not\Vdash \varphi.$



EF-inconsistency

$t \in T$ is EF-insonsistent in T

$\overset{\text{def}}{\Longleftrightarrow}$ $\exists \mathbf{E}_A\mathbf{F}\,\varphi.\ t \Vdash \mathbf{E}_A\mathbf{F}\,\varphi$, $\forall \pi$: A-path in T. $\pi_0 = t$, $\forall i.\ \pi_i \not\Vdash \varphi$.



Implementation using BDD

- Prepare enough BDD variables.
- Foreach BDD variable x, prepare a BDD variable x'.
- Foreach $\varphi \in \mathsf{Bas}(\varphi_0)$, allocate a BDD variable $e_{\mathsf{F}}(\varphi)$.
- For $t \in \mathsf{Type}$, $e(t) := \bigwedge\{\, e_{\mathsf{F}}(\varphi) \mid \varphi \in t\}$
  $\wedge \bigwedge\{\, \neg e_{\mathsf{F}}(\varphi) \mid \varphi \in \mathsf{Bas}(\varphi_0) \setminus t\}$
- For $T \subseteq \mathsf{Type}$, $e(T) := \bigvee\{e(t) \mid t \in T\}$.
- For $R \subseteq T \times T$
  $e(R) := \bigvee\{\, e(t_1) \wedge (e(t_2))' \mid (t_1, t_2) \in R \,\}$
  d': d with all BDD variables replaced with primed one.

Pseudo Code

```
BDD type, trans[], tabl[], u[];
Boolean satisfiable(φ) {
     type := ⋀{ e_F(φ) → ⋁{ e_M(φ, m) | m ∈ φ_M } | φ ∈ cl(φ_0)_EX }

     foreach (m ∈ M_0) {
          trans[m] := ⋀{ e_F(φ) → e'_F(φ_X) | φ ∈ cl(φ_0)_AX, m ∈ φ_M }
           ∧ ⋀{ e'_F(φ) → e_F(φ_X) | φ ∈ cl(φ_0)_AX, m̄ ∈ φ_M }
           ∧ ¬ ⋁{e_F(φ) ∧ ¬e_F(φ_R) ∧ e'_F(φ) ∧ ¬e'_F(φ_R) |
                                φ ∈ cl(φ_0)_AU, m ∈ φ_M, m̄ ∈ φ_M}
     }

     int k := 0;
     tabl[0] := type;
     repeat {
          tabl[k+1] := tabl[k] ∧ consis(k);
          k := k+1 ;
     } until (tabl[k] == tabl[k-1])
     if ( e_F(φ_0) ∧ tabl[k] != 0 ) { return true;} else { return false;}
}

BDD consis(k) { return consisEX(k) ∧ consisEU(k) ∧ consisAU(k); }

BDD consisEX(k) {
     return  ⋀{e_F(φ) ∧ e_M(φ, m) → ∃x⃗'(tabl[k]' ∧ e'_F(φ_X) ∧ trans[m]) |
                                φ ∈ cl(φ_0)_EX, m ∈ φ_M};
}
```

```
BDD consisEU(k) {
     return  ⋀{ e_F(φ) → fulfill(k, φ, stepEU) | φ ∈ Cl(φ_0)_EU };
}

BDD consisAU(k) {
     return  ⋀{ e_F(φ) → fulfill(k, φ, stepAU) | φ ∈ Cl(φ_0)_AU };
}

BDD fulfill(k, φ, step) {
     int j := 0;
     u[0] := tabl[k]∧e_F(φ_R);
     repeat {
          f[j+1] := f[j]  ∨ ( tabl[k]∧e_F(φ) ∧ ¬e_F(φ_R)∧step(k,φ,j) );
          j := j+1;
     } until (u[j] == u[j-1])
}

BDD stepEU(k, φ, j) { return ∃⃗x'(u[j]' ∧ ⋁{ trans[m] ∧ e_M(E_{φ_M} X φ, m) | m ∈ φ_M }) };

BDD stepAU(k, φ, j) {
    return
⋀{ e_F(ψ) ∧ e_M(ψ, m) → ∃⃗x'(u[j]' ∧ e'_F(ψ_X) ∧ trans[m]) | ψ ∈ Cl(φ_0)_EX, m ∈ φ_M ∩ ψ_M };
}
```

Advantages of BDD

An extreme example: $\varphi_0 := \bigwedge\{\mathbf{E}_a\mathbf{F}\, p_i \mid i = 1, \ldots, N\}$

- The size of Type: 2^N
- A naive implementation:
  - Creates all φ_0-types and the transition relation at the very first stage.
  - For each i, all nodes are visited to check EF-inconsistency.
  - $O(N \cdot 2^N)$ time.
- An implementation with BDD:
  - No need to create φ_0 types. The number of BDD variables is linear to N.
  - (For this particular φ_0,) the size of BDD is also linear to N during computation. $O(N)$ time.

Predicate Abstraction

Pointer System

Satisfiability Checking

Hybrid Temporal Logic

Hybrid Temporal Logic

- Formula: two kinds of atomic formulas:
  - Propositional Variables (the elements of AP)
  - Nominals (the elements of Nom)
- Kripke structure $K = (S, R, L)$:
  - $L : \mathsf{AP} \cup \mathsf{Nom} \to \mathscr{P}(S)$
  - For $n \in \mathsf{Nom}$, $L(n)$ is a singleton.

Satisfiability

Theorem (Sattler, Vardi)

The satisfiability problem of hybrid two-way modal μ-calculus is decidable. Its complexity is EXPTIME-complete.

U. Sattler and M. Y. Vardi. The Hybrid μ-Calculus. In *Proceedings of the International Joint Conference on Automated Reasoning*, LNAI Vol. 2083, pp. 76–91, 2001.

Simpler Algorithm for One-way CTL

$N :=$ the set of nominals appeard in φ_0.

Apply the satisfiability checking regarding nominals as ordinary propositional variables to get $T = T_0$.

EvForm $:= \{\varphi \in \mathsf{cl}(\varphi_0) \mid \varphi = \mathbf{E}_A\mathbf{F}\,\psi \text{ or } \varphi = \mathbf{A}_A\mathbf{F}\,\psi\}$

Guess two functions f and g:

- $f : N \to T$
- For $n \in N$, $n \in f(n)$.
- $g : N \times N \to \mathscr{P}(\mathsf{EvForm})$
- For $n_1, n_2 \in N$, $g(n_1, n_2) \subseteq \mathsf{Sat}(f(n_1))$
- $k \geq 2$, $n_1, \ldots, n_k \in N$, $n_1 = n_k$
  $\Longrightarrow \bigcap \{g(n_j, n_{j+1}) \mid j = 1, \ldots, k-1\} = \varnothing$

Intension: $f(n)$ is the unique φ_0-type such that $f(n) \Vdash n$. $\varphi \in g(n_1, n_2)$ means $f(n_1)$ can safely rely on $f(n_2)$ for the eventuality of φ.

Wlog, $\exists n \in N.\ f(n) \Vdash \varphi_0$.

$T' := \{t \in T_0 \mid n \in N, n \in t \implies t = f(n)\}$

For each $n \in N$, construct T_n.

1. Start with $T_n := T'$.
2. Remove $t \in T_n$ from T_n if t is either EX-inconsistent, EF-inconsistent, or AF-inconsistent in T_n with respect to n.
3. Repeat above as much as possible.
4. If $f(n)$ remains in T_n, then the process is all right for n.

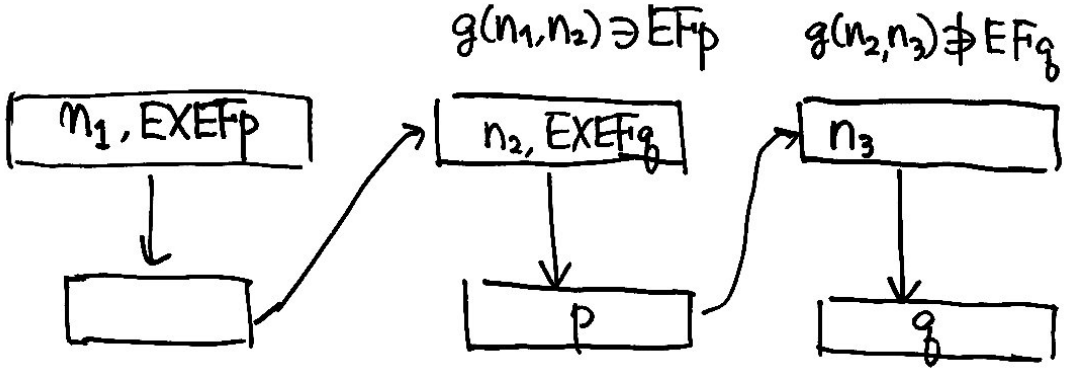
If the process is all right for all $n \in N$, φ_0 is satisfiable, otherwise the guess $(f,\ g)$ is not correct.

Consistency with Respect to n

Differences are:

- If $f(n) \neq f(n')$,
  $f(n')$ is always *-consistent with respect to n.
- When checking {EF, AF}-consistency with respect to n,
  - If $\varphi \in g(n, n')$, assume that $f(n') \Vdash \psi$ holds.
  - If $\varphi \notin g(n, n')$, assume that $f(n') \Vdash \psi$ does not hold.

  where $\varphi = \mathbf{E}_A\mathbf{F}\,\psi$ or $\mathbf{A}_A\mathbf{F}\,\psi$



Summary

- A verification method for software model checking on pointer systems based on predicate abstraction
- Temporal logics (+ α) as predicates
- Two key issues:
  - Calculation of precoditions
  - Satisfiability checking
- Implementation with BDD for satisfiability of temporal logic formula
- Hybrid temporal logic