Verification of Real Time Systems, CS5270 Guest Lecture

Real-time logic. Graph-theoretic analysis

Stefan Andrei
http://www.comp.nus.edu.sg/~andrei

Overview

- Real-time logic
- Counting true instances
- Incremental verification of the real-time systems specifications

PART 1. Real-time logic

References

- Chapter 6 of [Che2002] Cheng, A.M.K.: *Real-time systems. Scheduling, Analysis, and Verification*. Wiley-Interscience, 2002
- [JaM87] Jahanian, F., Mok, A.: A Graph-Theoretic Approach for Timing Analysis and its Implementation. *IEEE Transactions on Computers.* Vol. C-36, No. 8, 1987
- [RiC99] Rice, L.E.P., Cheng, A.M.K.: Timing Analysis of the X-38 Space Station Crew Return Vehicle Avionics. *Proceedings of the 5-th IEEE-CS Real-Time Technology and Applications Symposium*, pp. 255-264, 1999

Specification of real-time systems

- Structurally and functionally specification (how the real-time system components work as well as their functions and operations):
  - Mechanical components
  - Electrical components
  - Electronic components
- Behavioral specification
  - Sequences of events in response to actions

Example: real-time anti-lock braking system in an automobile

- Structural-functional specification refers to:
  - Braking system components and sensors
  - How they are interconnected, and
  - How the actions of each component affects the other
  - *Example: how to connect the wheel sensors to the central decision-making computer that controls the brake mechanism.*
- Behavioral specification refers to:
  - Events and effects
  - *Example: when the wheel sensors detect wet road conditions, the decision-making computer will instruct the brake mechanism to pump the brakes at a higher frequency within 100ms.*

Timing Constraints

- Behavioral specification without the complexity of the structural specification often suffices;
- We restrict the specification language to handle only timing relations.

Specification and safety assertion

- An implementation of a real-time system is built from the structural-functional specification;
- An implementation is **correct** (faithful) if
  - the behavioral specification (denoted as SP) implies safety assertions (denoted as SA).
  - In other words, we have to check whether SP → SA is a theorem or not.

Verification of Timing Properties

- In checking *SP* → *SA*, we may have the cases:
  - (**safe**) *SA* is a theorem derivable from *SP*;
  - (**inherently unsafe**) *SA* is unsatisfiable with respect to *SP*;
  - (**safe if additional constraints are added**) the negation of *SA* is satisfiable under certain conditions.

Event-action model

- [Hen80] Heninger, K.L.: Specifying Software Requirements for Complex Systems: New Techniques and Their Applications. *IEEE Trans. Software Engineering.* vol. SE-6, no. 1 (1980) 2-13
- Heninger captured the data dependency and temporal ordering of computational actions that must be taken in response to events in a real-time application.

Concepts of event-action model

- Syntax of Actions:
  - *<Action> = <primitiveAction> |*
    <Action> ; <Action> |
    <Action> "||" <Action>
  - Examples:
    - *TRAIN_APPROACH; DOWN_GATE* is sequential execution of two primitive actions, i.e. a composite action;
    - *DOWN_GATE* || *RING_BELL* is parallel execution of two primitive actions, i.e., a composite action;
- State predicate: *Event x Time → Bool*
  - Example: *GATE_IS_DOWN* is *true* if the gate is in the down position

Concepts of event-action model (cont)

- Event
  - External: *APPLY_BRAKE*
  - Start: the start of *DOWN_GATE*
  - Stop: the end of *DOWN_GATE*
  - Transition: *GATE_IS_DOWN* becomes *true* when gate is moved down.
- Timing constraint (absolute timing of system events)
  - Example: the timing difference between the start and the end of *DOWN_GATE* may take at least *15* seconds.

Real-Time Logic (RTL)

- Motivation: event-action model cannot be easily manipulated by a computer ([JaM87]);
- RTL = first-order logic with special features to capture the (absolute) timing requirements;
- RTL is based on the event-action model;
- *@:: Event x Occurrence → Time*, where *Occurrence=Nat-{0}* and *Time=Nat*.
- *Semantics:*
  - *@(e, i) = t* means the *i-th* occurrence of event *e* occurs at time *t*.
  - $\forall e \in Event$, $\forall i \in Occurrence$, *@(e,i) < @(e,i+1)* if *@(e,i+1)* is defined.

Real-Time Logic (cont)

- Three types of RTL constants:
  - Actions: a subaction B_i of a composite action A is denoted by $A.B_i$
  - Events constants are temporal markers
    - External Events: Ω*event-name*
    - Start Events: $\uparrow$*event-name*
    - Stop Events: $\downarrow$*event-name*
    - Transition Events: change in certain attributes of the system state;
  - Integers: used for timing constraints.

Example of a real-time system: railroad crossing

- Structural-functional specification:
  - Field measurements, mechanical characteristics of the train, train sensor, gate controller, gate;
- The goal of gate controller: when train is crossing the intersection, no car is on the intersection;
- Simplified goal (safety assertion): when train is crossing, the gate is in the down position.

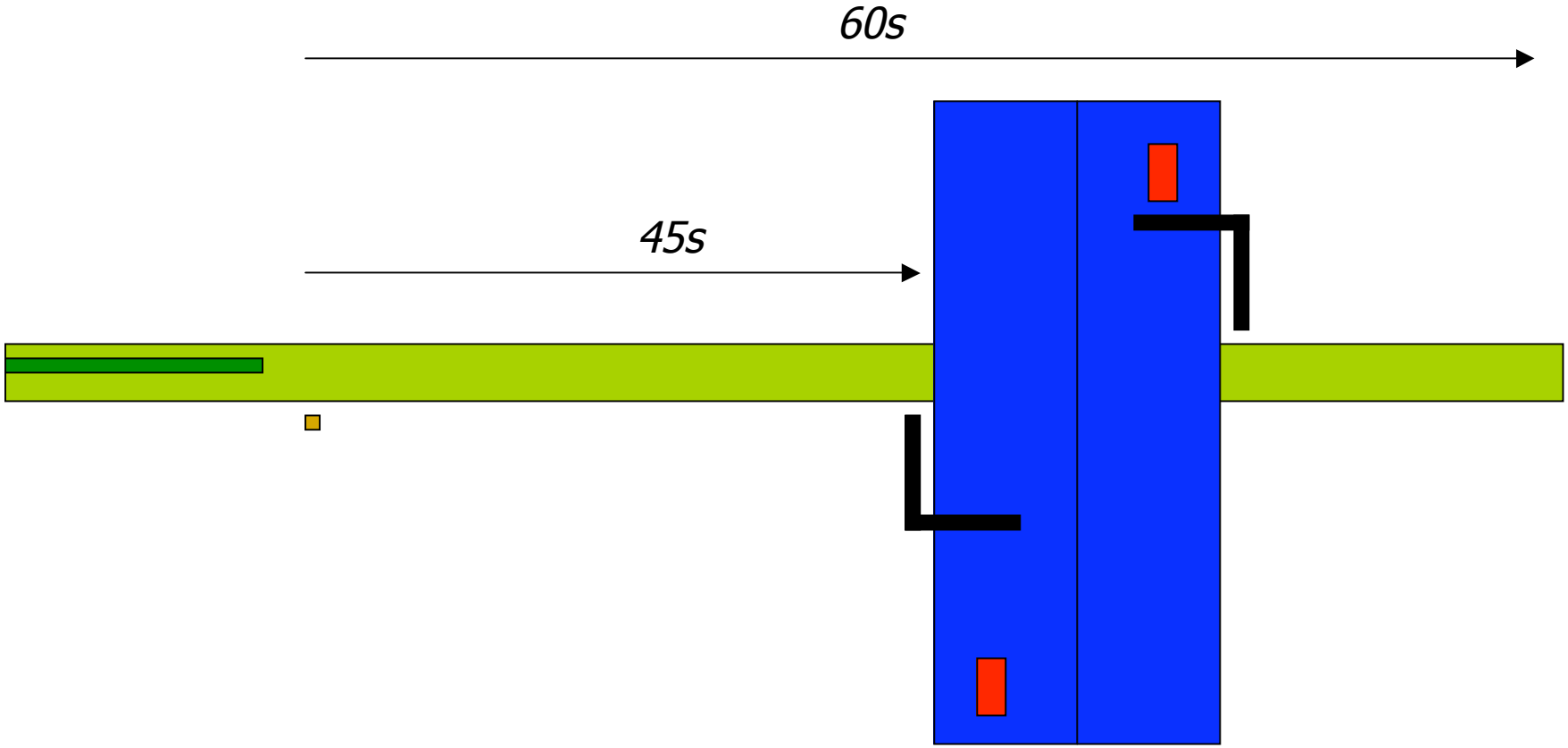
Behavioral Specification (English)

- When train approaches sensor, a signal will initiate the lowering of gate, **and**
- Gate is moved to down position within *30s* from being detected by the sensor, **and**
- The gate needs at least *15s* to lower itself to the down position.

Safety Assertion (English)

- **If**
  - train needs at least *45s* to travel from sensor to the railroad crossing, **and**
  - the train crossing is completed within *60s* from being detected by sensor,
- **then**
  - we are assured that at the start of the train crossing, gate has moved down **and**
  - that the train leaves the railroad crossing within *45s* from the time the gate has completed moving down.

Railroad crossing (animation)



Behavioral Specification (RTL)

- $\forall x \; (\; @(TrainApproach, x) \leq @(\uparrow DownGate, x) \wedge$
  $@(\downarrow DownGate, x) \leq @(TrainApproach, x) + 30$
  $)$

- $\forall y \; (@(\uparrow DownGate, y) + 15 \leq @(\downarrow DownGate, y)$
  $)$

Safety Assertion (RTL)

- $\forall t\ \forall u\ ($
  $@(TrainApproach, t) + 45 \leq @(\uparrow TrainCrossing, u) \wedge$
  $@(\downarrow TrainCrossing, u) < @(TrainApproach, t) + 60 \rightarrow$
  $@(\uparrow TrainCrossing, u) \geq @(\downarrow DownGate, t) \wedge$
  $@(\downarrow TrainCrossing, u) \leq @(\downarrow DownGate, t) + 45$
  $)$
- In a simplified version, we can consider $u = t$

Presburger Arithmetic Formulae

- Is built from constraints using: $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$, (,)
- A constraint is a series of expression lists, connected with: $=$, $!=$, $<$, $\leq$, $>$, $\geq$
- If *var* is a variable, *int* is an integer, and e, e_1, e_2 are expressions, then *var*, *int*, e, *int* e, e_1+e_2, e_1-e_2, $int*e_2$, $-e$ and (e) are expressions;
- Each $@(e,i)$ is replaced by a function $f_e(i)$, where e is an event and i is an integer or an variable.

Presburger Arithmetic Formulae

- Specification (*SP*)
  - $\forall x\ (f(x) \leq g_1(x) \wedge g_2(x) \leq f(x) + 30)$
  - $\forall y\ (g_1(y) + 15 \leq g_2(y))$
- Safety Assertion (*SA*)
  - $\forall t\ \forall u\ ($

    $f(t) + 45 \leq h_1(u) \wedge h_2(u) < f(t) + 60 \rightarrow$

    $g_2(t) \leq h_1(u) \wedge h_2(u) \leq g_2(t) + 45$

    $)$

Restricted RTL formulas

- The problem *SP* → *SA* is in general undecidable for the full set of RTL formulas.
- [JaM87] motivated that RTL formulas of many real-time systems:
  - Consist in arithmetic inequalities involving two terms and an integer constant in which a term is either a variable or a function (difference constraints).
  - Do not contain arithmetic expressions that have a function taking an instance of itself as an argument.
  - [WaM94] Wang, F., Mok, A. K.: RTL and Refutation by Positive Cycles. *Proceedings of Formal Methods Europe Symposium*, 873, Lecture Notes in Computer Science, pp. 659-680, 1994.

The path-RTL formulas [JaM87, WaM94]

- The general form of path-RTL formulas:

functionOccurrence ± *integerConstant* ≤ *functionOccurrence*

- Industrial real-time systems:
  - Railroad crossing [JaM87], [JaS88], [Che2002]
  - Moveable control rods in a reactor [JaM87]
  - Boeing 777 Integrated Airplane Information Management System [MTR96]
  - X-38, an autonomous spacecraft build by NASA [RiC99]

References

- This kind of Presburger Formulas was studied in many papers, such as:
  - [Pug94] Pugh, W.: Counting Solutions to Presburger Formulas: How and Why. *PLDI'94. ACM SIGPLAN* 94-6/94 (1994) 121-134
  - [LLP97] Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction. *IEEE RTSS'97, CS Press* (1997) 14-24
  - [Min2001] Miné, A.: A New Numerical Abstract Domain Based on Difference-Bound Matrices. Program as Data Objects II. *LNCS 2053* (2001) 155-172

Behavioral Specification implies Safety Assertions

- $SP \rightarrow SA$ is tautology iff $\neg(SP \rightarrow SA)$ is unsatisfiable;
- $\neg(SP \rightarrow SA) \equiv \neg(\neg\, SP \vee SA) \equiv SP \wedge \neg\, SA$
- Therefore, $SP \rightarrow SA$ is tautology iff $SP \wedge \neg\, SA$ is unsatisfiable.

Clausal Form

- Specification (*SP*):
  - $\forall x \forall y\ (f(x) \le g_1(x) \wedge g_2(x) - 30 \le f(x) \wedge g_1(y) + 15 \le g_2(y))$
- **Negation of Safety Assertions (¬*SA*):**

$\exists t \exists u\ (f(t) + 45 \le h_1(u) \wedge h_2(u) < f(t) + 60 \wedge (h_1(u) < g_2(t) \vee g_2(t) + 45 < h_2(u)));$

  - **Skolem normal form of RTL formulas [*T/t*][*U/u*]:**

$f(T) + 45 \le h_1(U) \wedge h_2(U) - 59 \le f(T) \wedge (h_1(U) + 1 \le g_2(T) \vee g_2(T) + 46 \le h_2(U))$

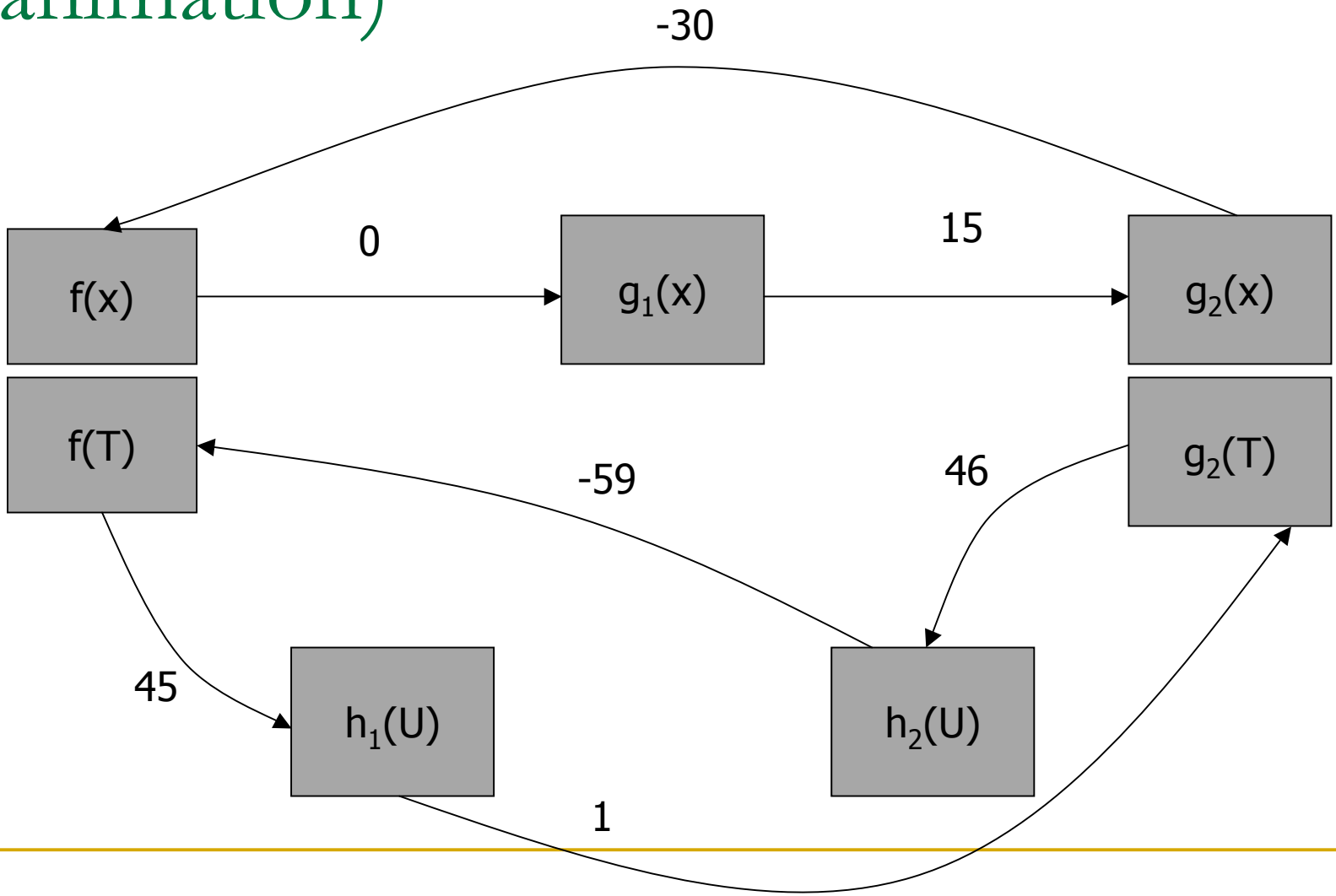
[JaM87] Strategy

- F – the initial RTL Formula;
- F' – the corresponding Presburger Formula;
- F'' – the Clausal Formula corresponding to $SP \wedge \neg SA$, that is: $C_1 \wedge C_2 \wedge \ldots \wedge C_n$, where $C_i = L_{i,1} \vee L_{i,2} \vee \ldots \vee L_{i,n}$ and each $L_{i,j}$ has the general form: $v_1 \pm l \leq v_2$, l being a positive integer constant.

Constraint Graph Construction

- For each literal $v_1 \pm I \leq v_2$, we construct a node labeled v_1, a node labeled v_2, and an edge $<v_1, v_2>$ with weight $\pm I$ from node v_1 to node v_2;
- If the constraint graph contains a cycle with positive weight, then *F"* is unsatisfiable.

Railroad Crossing Constraint Graph (animation)



Positive cycles lead to unsatisfiability

- Let $X_{i,1}, X_{i,2}, \ldots, X_{i,n^i}$ the *i*-th positive cycle. The sum of weights of edges is positive, so $P_i = X_{i,1} \wedge X_{i,2} \wedge \ldots \wedge X_{i,n^i}$ is unsatisfiable ([JaM87]); therefore, $\neg P_i$ is tautology;
- Therefore, *F''* is (un)satisfiable iff $F'' \wedge \{\neg P_i \mid$ *for all positive cycle i}* is (un)satisfiable;

Railroad Crossing Satisfiability

- $A = f(x) \leq g_1(x)$
- $B = g_2(x) - 30 \leq f(x)$
- $C = g_1(y) + 15 \leq g_2(y)$
- $D = f(T) + 45 \leq h_1(U)$
- $E = h_2(U) - 59 \leq f(T)$
- $F = h_1(U) + 1 \leq g_2(T)$
- $G = g_2(T) + 46 \leq h_2(U)$
- *F''* has the positive clauses: *A, B, C, D, E, F v G*

Positive cycles lead to unsatisfiability (cont)

- A positive/negative clause contains only positive/negative literals (for example, $F \vee G$ is a positive clause, whereas $\neg B \vee \neg D \vee \neg F$ is a negative clause);
- *F"* contains positive clauses corresponding to all edges, and negative clauses corresponding to a positive cycle;
- The CNF satisfiability is NP-complete even if each clause is positive or negative ([JaM87]).

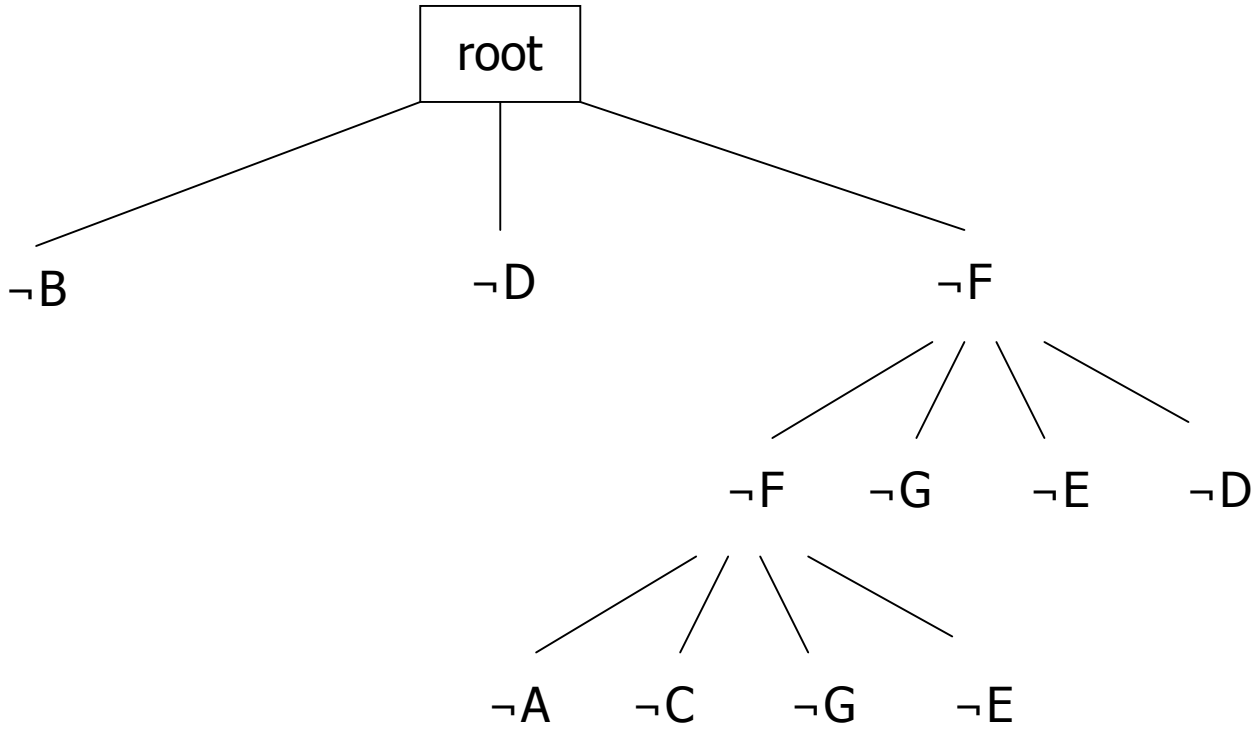
Railroad Crossing Satisfiability (cont)

- Three positive cycles in the constraint graph imply that *F''* has the negative clauses:
  - $\neg B \vee \neg D \vee \neg F$
  - $\neg F \vee \neg G \vee \neg E \vee \neg D$
  - $\neg A \vee \neg C \vee \neg G \vee \neg E$

Search tree

- The Resolution Method works for *F”*, but it is not so efficient;
- More efficient, [JaM87] transformed the set of negative clauses from conjunctive normal form into disjunctive normal form;
- This corresponds to a tree, where each leaf will be checked to at least one positive clause or by itself;
- By considering the negation of unitary clauses as early as possible, the strategy of building the tree can be improved even more.

Search tree for Railroad Crossing



- *F''* = *A* ∧ *B* ∧ *C* ∧ *D* ∧ *E* ∧ (*F* ∨ *G*) ∧ (¬*B* ∨ ¬*D* ∨ ¬*F*) ∧ (¬*F* ∨ ¬*G* ∨ ¬*E* ∨ ¬*D*) ∧ (¬*A* ∨ ¬*C* ∨ ¬*G* ∨ ¬*E*)
- *F''* is unsatisfiable, so *SP* → *SA* is tautology;

Conclusions of Part 1

- So far, the presentation was based on [JaM87] and [Che2002];
- We discuss another strategy based on counting the number of true instances of F''. This will tell us how "far away" is the current specification from satisfying the safety assertion;
- The addition of a new positive cycle may result from a modification of the specification and/or safety assertions. This is useful for incremental debugging, in which bugs in problematic areas are fixed one at a time until the system is correct.
- Special thanks to Professor Albert M. K. CHENG.

PART 2. Counting true instances

Counting true instances

- [Iwa89] Iwana, K.: CNF Satisfiability Test by Counting and Polynomial Average Time. *Siam J. Comput.* 18, No. 2 (1989) 385-391
- [Dub91] Dubois, O.: Counting the number of solutions for instances of satisfiability. *Theoretical Computer Science.* 81 (1991) 49-64
- [Tan91] Tanaka, Y.: A dual algorithm for the satisfiability problem. *Information Processing Letters* 37 (1991) 85-89
- [And95] Andrei, S.: The Determinant of the Boolean Formulae, *Analele Universitatii Bucuresti, Informatica* (1995) 83-92
- [And99] Andrei, S.: Weak Equivalence in Propositional Calculus. *Proceedings of European Summer School on Logic, Language and Information*, pp.79-89, Universiteit Utrecht, August 1999

Notations ([And95])

- $F|_V=\{C_1,..,C_l\}$ a clausal formula over $V=\{A_1,..,A_n\}$.
- Example: $F|_V=\{C_1,C_2,C_3,C_4\}$, $V=\{p,q,r\}$, where $C_1=\{p, \neg r\}$, $C_2=\{\neg q, r\}$, $C_3=\{q, \neg r\}$, $C_4=\{\neg p, q, r\}$.
- If $C_1',\ldots,C_s' \in F|_V$ and $s \leq l$, then:
  - $m(C_1',\ldots,C_s')$= number of atomic formulae from V which do not occur in $C_1' \cap \ldots \cap C_s'$. For example: $m(C_1)=1$, $m(C_2)=1$, $m(C_3)=1$, $m(C_4)=0$, $m(C_1,C_2)=0$, $m(C_1,C_3)=0$, $m(C_1,C_4)=0$, etc.

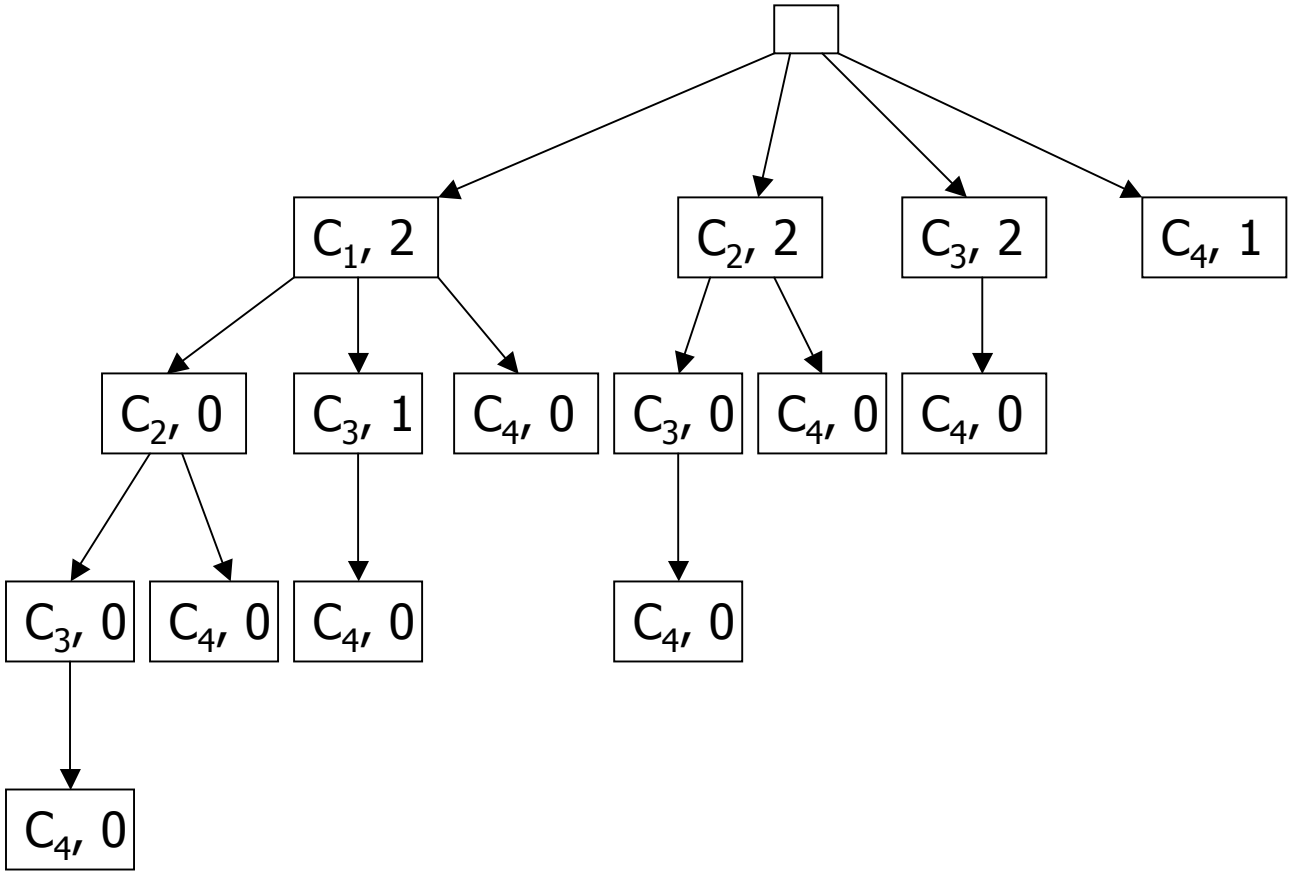
Dual Resolution Theorem ([And95])

- $dif(C_1', \ldots, C_s')$=
  - 0 if $\exists\, i, j \in \{1,\ldots,s\}$, $i \neq j$, $\exists\, L$ literal such that $L \in C_i'$ and $\neg L \in C_j'$
  - $2^{m(C1', \ldots, Cs')}$ otherwise
  - Example: $dif(C_1)=2$, $dif(C_2)=2$, $dif(C_3)=2$, $dif(C_4)=1$, $dif(C_1,C_2)=0$, $dif(C_1,C_3)=1$, $dif(C_1,C_4)=0$, etc.
- $det(F|_V)=2^n - \sum_{s=1}^{l} (-1)^{s+1} * \sum_{1 \le i1 < \ldots < is \le l} dif(C_1', \ldots, C_s')$ is called the **determinant** of $F|_V$ ([And95]).
- **Theorem.** $F|_V$ has $det(F|_V)$ truth assignments. So, $F|_V$ is satisfiable iff $det(F|_V) \neq 0$.
- Example: $det(F|_V)=2^3-(2+2+2+1-1)=2$, so $F|_V$ is satisfiable having 2 truth assignments.

Ordered labelled standard tree

- A node in OLST is labelled with the clause $C_{i,k}$ and $dif(C_{i,1}, .., C_{i,k})$, where $C_{i,1}, .., C_{i,k}$ is the sequence of nodes from the root $C_{i,1}$ to the current node $C_{i,k}$
- The total number of nodes is 2^l.

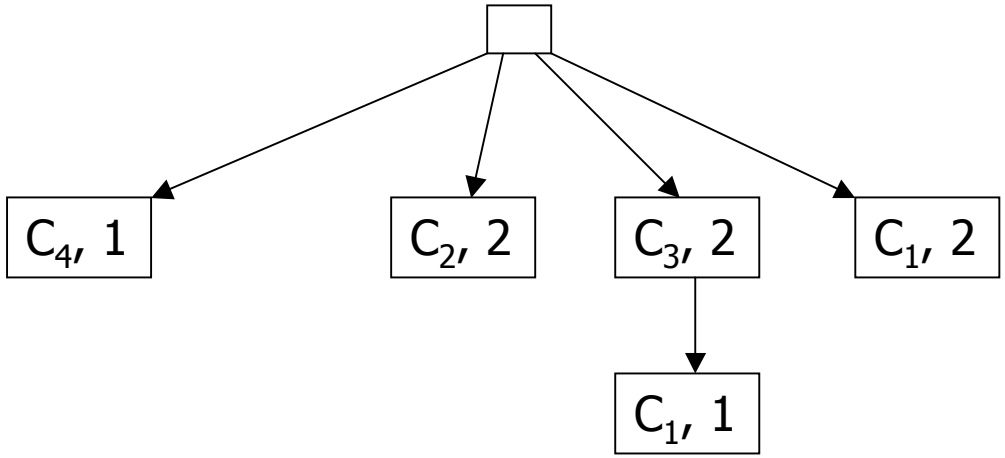
Ordered labelled standard tree (cont)



- The total number of nodes is 2^4=16 (there are 4 clauses).
- $det(F|_V)=2^3-(2+2+2+1)+(0+1+0+0+0+0)-(0+0+0+0)+0=2.$

Ordered labelled reduced tree (cont)

- If *dif*(C_1',…,C_s')=*0* then *dif*(C_1',…,C_s',C_{s+1}')=0.



- The nodes labelled with *dif(...)=0* need not be generated!
- Now, the total number of nodes is 6.
- *det*($F|_V$)=2^3-(1+2+2+2)+(1)=2.

PART 3. Incremental verification of the real-time systems specifications

Our Incremental Approach for Systematic Debugging [AnC04]

Input: SP, SA such that $\neg SA$ is satisfiable;
Output: SP_{new}, SA_{new} such that the system is safe;
Method:

1. $k = 1$; $SP_1 = SP$; $SA_1 = SA$;
2. while ($SP_k \rightarrow SA_k$ is not a tautology) {
3. let SP_{new} and SA_{new} be new constraints;
4. $SP_{k+1} = SP_k \oplus SP_{new}$;
5. $SA_{k+1} = SA_k \oplus SA_{new}$;
6. $k = k + 1$; }
7. $SP_{new} = SP_k$; $SA_{new} = SA_k$;

Past Work [AnC04]

- The satisfiability of $SP_{k+1} \rightarrow SA_{k+1}$ is expressed incrementally from the satisfiability of $SP_k \rightarrow SA_k$
- The **manual** debugging from step 3 is correlated with the satisfiability of $SP_k \rightarrow SA_k$
- We use #SAT problem rather than SAT problem:
  - To know how "far away" is *SP* from satisfying *SA*;
  - The modification of *SP* and/or *SA* is useful for incremental debugging, in which bugs are fixed one at a time until the system is correct.

- Andrei, S., Chin, W.-N.: Incremental Satisfiability Counting for Real-Time Systems. *The 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*, Toronto, Canada, 25 May – 28 May, 482-489, 2004

Past Work [ACCL05]

- The debugging from step 3 is done **systematically**, not manually.
- Since the industrial real-time systems may have large specifications, it is impracticable for the designer to find the proper missing or wrong constraints.
- Efficient Java implementation of systematic debugging (http://www.comp.nus.edu.sg/~andrei/SDRTL/). Examples of real-time systems have also been successfully tested by SDRTL.
- We simulated a real-life scenario, supposing that the designer may forget to include some constraints or may give some incorrect constraints.

- Andrei, S., Chin, W.-N., Cheng, A.M.K., Lupu, M.: Systematic Debugging of Real-Time Systems based on Incremental Satisfiability Counting. *The 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'05)*, San Francisco, United States, 7 March - 10 March, 10 pages, 2005

Re-design of Railroad Example

- We consider new events and new constraints.
- We add to *SP*:
  - (English) A car needs at most *10* seconds to cross the railroad;
  - *(RTL)* $@(\downarrow CarCrossing, z) - 10 \leq @(\uparrow CarCrossing, z)$
- We add to *SA*:
  - (English) If the train starts to cross the railroad crossing, there is no car crossing in the last *5* seconds;
  - *(RTL)* $@(\downarrow CarCrossing, v) + 5 \leq @(\uparrow TrainCrossing, u)$

Re-design of Railroad Example - Presburger Arithmetic Formulas

- SP_1:
  - $\forall x\ (f(x) \le g_1(x) \wedge g_2(x) \le f(x) + 30)$
  - $\forall y\ (g_1(y) + 15 \le g_2(y))$
  - $\forall z\ (\eta_2(z)\text{-}10 \le \eta_1(z))$
- SA_1:
  - $\forall t\ \forall u\ \forall v\ ($

    $f(t) + 45 \le h_1(u) \wedge h_2(u) < f(t) + 60 \rightarrow$

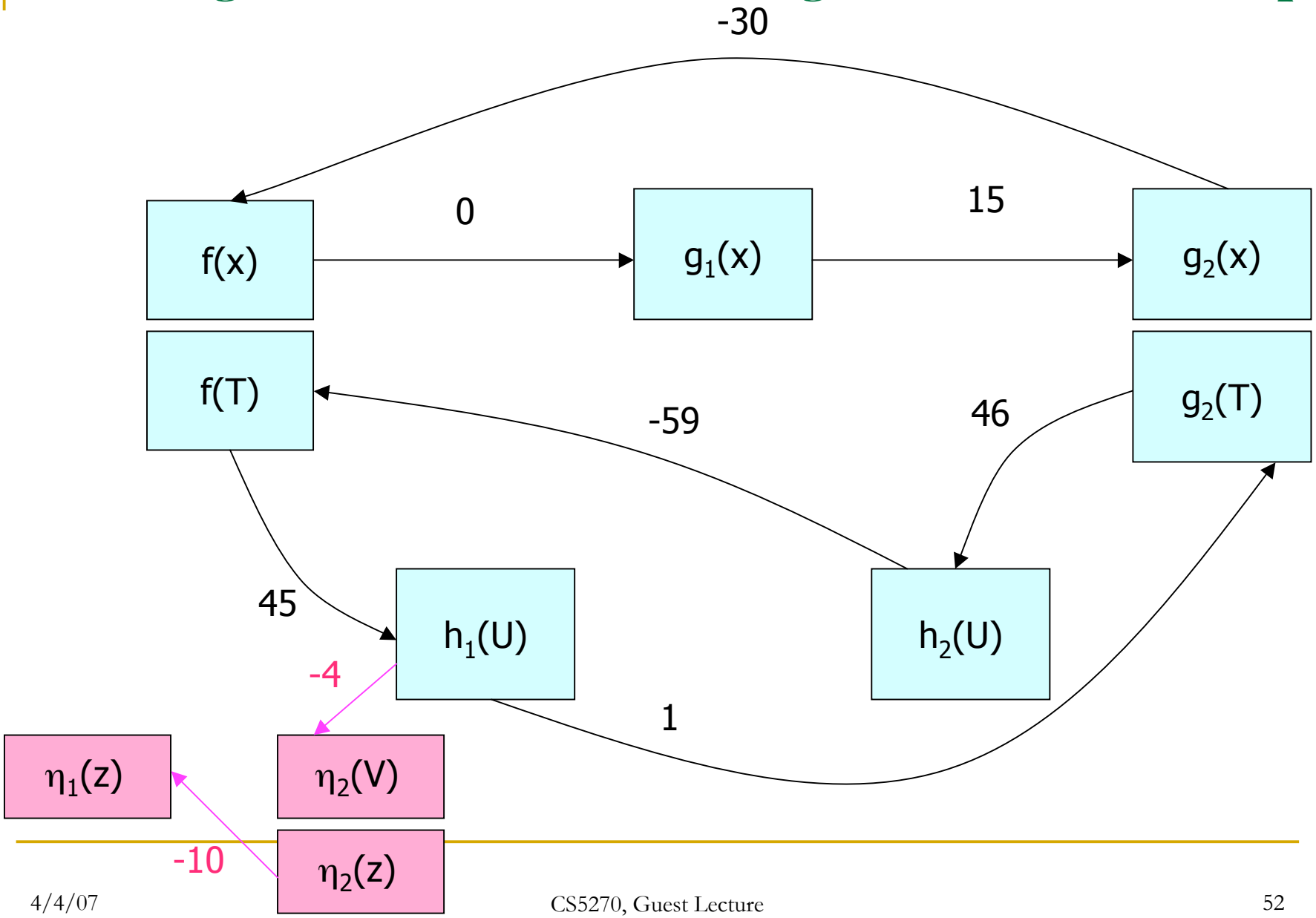
    $g_2(t) \le h_1(u) \wedge h_2(u) \le g_2(t) + 45 \wedge \eta_2(v)+5 \le h_1(u)$

    $)$

Re-design of Railroad Example – Clausal Form

- SP_1:
  - $\forall x \forall y \forall z (f(x) \leq g_1(x) \wedge g_2(x) - 30 \leq f(x) \wedge g_1(y) + 15 \leq g_2(y) \wedge \eta_2(z) - 10 \leq \eta_1(z))$
- Negation of Safety Assertions ($\neg SA_1$):
  - $f(T) + 45 \leq h_1(U) \wedge h_2(U) - 59 \leq f(T) \wedge$
    $(h_1(U) + 1 \leq g_2(T) \vee g_2(T) + 46 \leq h_2(U) \vee h_1(U) - 4 \leq \eta_2(V))$

Re-design of Railroad Crossing - Constraint Graph



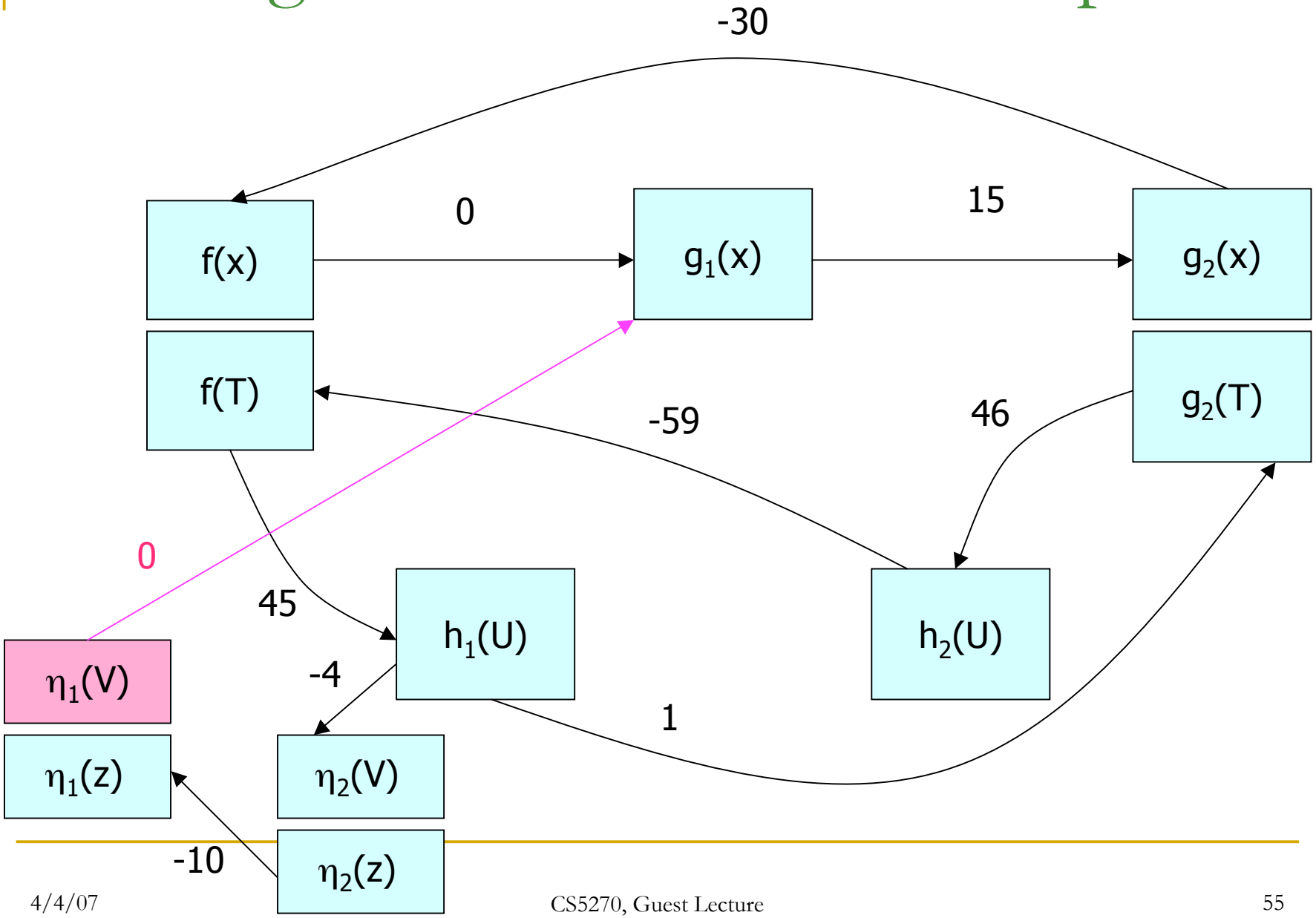
Re-design of Railroad Crossing – PF_1

- Literals:
  - $A = f(x) \leq g_1(x)$, $B = g_2(x) - 30 \leq f(x)$, $C = g_1(y) + 15 \leq g_2(y)$,
  - $D = f(T) + 45 \leq h_1(U)$, $E = h_2(U) - 59 \leq f(T)$,
  - $F = h_1(U) + 1 \leq g_2(T)$, $G = g_2(T) + 46 \leq h_2(U)$,
  - $H = \eta_2(z) - 10 \leq \eta_1(z)$, $I = h_1(U) - 4 \leq \eta_2(V)$
- PF_1 has the positive clauses: A, B, C, D, E, H, $F \vee G \vee I$
- PF_1 has the negative clauses:
  - $\neg B \vee \neg D \vee \neg F$
  - $\neg F \vee \neg G \vee \neg E \vee \neg D$
  - $\neg A \vee \neg C \vee \neg G \vee \neg E$

Re-design of Railroad Crossing – Satisfiability

- We get $det(PF_1) = 3 > 0$, hence PF_1 is satisfiable;
- So, the real-time system may be unsafe;
- Is it necessary to have at least one more positive cycle containing *H* and *I;*
- Going back to SA_1, we add:
  - (English) If the gate starts to go down, then no car will start to cross the railroad
  - (RTL) $@(\uparrow CarCrossing, v) \leq @(\uparrow DownGate, t)$
  - (Presburger) $\eta_1(v) \leq g_1(t)$

Re-design - New Constraint Graph



Re-design of Railroad Crossing – PF_2

- Literals:
  - $A = f(x) \leq g_1(x)$, $B = g_2(x) - 30 \leq f(x)$, $C = g_1(y) + 15 \leq g_2(y)$,
  - $D = f(T) + 45 \leq h_1(U)$, $E = h_2(U) - 59 \leq f(T)$, $F = h_1(U) + 1 \leq g_2(T)$,
  - $G = g_2(T) + 46 \leq h_2(U)$, $H = \eta_2(z) - 10 \leq \eta_1(z)$,
  - $I = h_1(U) - 4 \leq \eta_2(V)$, $J = \eta_1(V) \leq g_1(T)$
- PF_2 has the positive clauses: A, B, C, D, E, H, J, $F \vee G \vee I$
- PF_2 has the negative clauses:
  - $\neg B \vee \neg D \vee \neg F$
  - $\neg F \vee \neg G \vee \neg E \vee \neg D$
  - $\neg A \vee \neg C \vee \neg G \vee \neg E$
  - $\neg I \vee \neg H \vee \neg J \vee \neg C \vee \neg B \vee \neg D$

Incremental #SAT

- **Problem:** Knowing the number of true instances of *PF*, what is the number of true instances of *PF*u{C}, for any arbitrary clause C?
- **Incremental computation:** we use $det_V(PF_1)$ to get $det_V(PF_2)$, without recomputing the common parts of PF_1 and PF_2

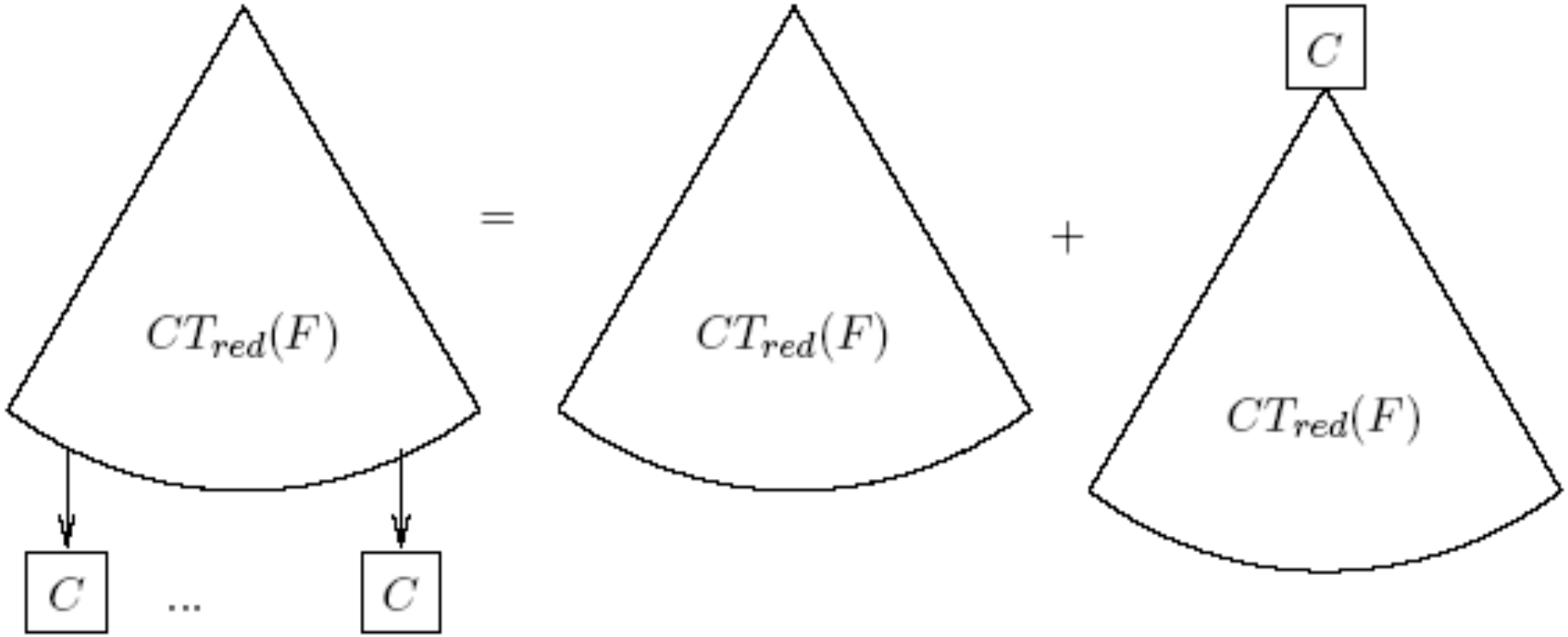
The Increment of a Clausal Formula

- **Definition:** Given $PF=\{C_1,\ldots,C_l\}$ over V and C an arbitrary clause, then

  $inc_V(C,PF)=\sum_{s=0}^{l}(-1)^{s+1} * \sum_{1\le i1<\ldots<is\le l} dif_V(C,C_{i1},\ldots,C_{is})$

  is called the **increment** of PF with C over V.
- $inc_V(C, PF)$ *is* represented as an **ordered labeled reduced clausal incremental tree**: $CIT_{red}(C,PF)$.

The Incremental Splitting of the Clausal Tree



Incremental Computing is Optimal

- **Theorem:** Let $PF=\{C_1,\ldots,C_l\}$ be a clausal formula and $PF'=\{C_{l+1},\ldots,C_{l+k}\}$. Then:
  - $det_V(PF \cup PF') = det_V(PF) + inc_V(C_{l+1}, PF) + inc_V(C_{l+2}, PF \cup \{C_{l+1}\}) + .. + inc_V(C_{l+k}, PF \cup \{C_{l+1},.., C_{l+k-1}\})$
  - $N'=N+N_{l+1}+\ldots+N_{l+k}$, where N', N, N_{l+1}, …, N_{l+k} are the number of nodes of the reduced clausal trees of $det_V(PF \cup PF')$, $det_V(PF)$, $inc_V(C_{l+1},PF),\ldots, inc_V(C_{l+k},PF \cup \{C_{l+1},\ldots, C_{l+k-1}\})$.
- Incremental computing is optimal.

Properties of $det_V(PF)$ and $inc_V(C,PF)$

- Let $PF = \{C_1, \ldots, C_l\}$ be a clausal formula over V. Then:
  - if A is an atomic variable, $A \notin V$, then $inc_{VU\{A\}}(\{A\}, PF) = inc_{VU\{A\}}(\{\neg A\}, PF) = -det_V(PF)$;
  - If V' is an alphabet such that $V \subseteq V'$ and C an arbitrary clause over V, then $det_{V'}(PF) = 2^{|V'|-|V|} * det_V(PF)$ and $inc_{V'}(C,PF) = 2^{|V'|-|V|} * inc_V(C,PF)$.

Satisfiability of the Railroad Crossing

- $PF_1 = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{H\}, \{F,G,I\}, \{\neg B, \neg D, \neg F\}, \{\neg F, \neg G, \neg E, \neg D\}, \{\neg A, \neg C, \neg G, \neg E\}\}$ over $V_1 = \{A, B, C, D, E, F, G, H, I\}$
- $PF_2 = PF_1 \cup \{\{J\},\{\neg I, \neg H, \neg J, \neg C, \neg B, \neg D\}\}$ over $V_2 = V_1 \cup \{J\}$
- $det_{V2}(PF_2) = det_{V2}(PF_1) + inc_{V2}(\{J\}, PF_1) + inc_{V2}(\{\neg I, \neg H, \neg J, \neg C, \neg B, \neg D\}, PF_1 \cup \{\{J\}\}) =$

  $= 2 * det_{V1}(PF_1) - det_{V1}(PF_1) - 3 = 3 - 3 = 0$
- The real-time system is safe now!

Experimental results

- Denote
  - $CT_{red}(F \cup \{C_{l+1}\} \cup \{C_{l+2}\})$ by CT_{red}^{new}
  - $CIT_{red}(C_{l+1}, F)$ by CIT_{red}^{1}
  - $CIT_{red}(C_{l+2}, F \cup \{C_{l+1}\})$ by CIT_{red}^{2}
  - n the number of variables, l the number of clauses

(n, l)	CT_{red}^{new}		$CT_{red}(F)$		CIT_{red}^{1}		CIT_{red}^{2}	
	Number of nodes	Time (sec.)	Number of nodes	Time (sec.)	Number of nodes	Time (sec.)	Number of nodes	Time (sec.)
(10, 20)	28831	0.16	12655	0.06	1760	0.01	14416	0.05
(15, 25)	70255	0.37	17799	0.13	17800	0.11	34656	0.21
(20, 40)	136714	3.32	99671	2.48	19832	0.39	17211	0.41
(25, 45)	78468	2.18	49800	1.50	6258	0.16	22410	0.71
(30, 60)	178531	7.70	141663	6.03	12700	0.83	24168	1.28
(40, 75)	150693	11.64	111837	8.77	13667	1.42	25189	2.19
(50, 100)	312276	39.26	268790	33.57	3701	0.67	39785	5.66

Related Work: Incremental Approaches

- An incremental positive cycle detection algorithm [MTR96] is also based on the constraint-graph technique and uses an algorithm for single source with positive weight in the graph.
- An incremental algorithm for model checking using transition systems in the alternation-free fragment of the modal μ-calculus was presented in [SoS94].
- Instead, our incremental approach is applied to propositional formulas.

History of SAT and #SAT problems

▪ The SAT problem ▪ [Cook, 1971]	▪ The #SAT problem ▪ [Valiant, 1979]
▪ The incremental SAT problem ▪ [Hooker, 1993]	▪ The incremental #SAT problem ▪ [Andrei & Chin, 2004]

Automatic Debugging [ACCL06]

- **Motivation:** The embedding and the integration of our debugger in autonomous systems where the specifications must meet timing constraints, but without human interaction.
- The idea is to consider in advance all the necessary information such as the designer's guidance.
- Efficient Java implementation for automatic debugging.

- [ACCL06] Andrei, S., Chin, W.-N., Cheng, A.M.K., Lupu, M.: Automatic Debugging of Real-Time Systems based on Incremental Satisfiability Counting. *IEEE Transaction on Computers*, vol. 55(7), pp. 830-842 (2006) Selected as July issue's Feature Article for 'hot' topic and fast publication.

Optimization of Specifications[AnC06']

- **Motivation:** After verifying *SP* -> *SA*, and the system implementing *SP* is deployed, performance changes as a result of power-saving, faulty components, and cost-saving in the processing platform for the tasks specified in *SP*.
- This leads to a different but related *SP*.
- It is desirable to determine an optimal *SP* with the slowest possible computation times for its tasks such that *SA* holds.
- The idea: relax *SP* and tighten *SA* such that *SP* -> *SA* is still a theorem.

- [AnC06'] Andrei, S., Cheng, A.M.K.: Optimization of Real-Time Systems Timing Specifications. *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2006)*, 7 pages, IEEE Computer Society, Sydney, August 16-18, 2006

Extension of Path-RTL [AnC06"]

- We shall present an extension of the path RTL class by allowing inequalities like

  $$\forall i \; @(e_1, i) + @(e_2, i) \leq k$$

  and

  $$\forall i \; @(e_1, i) + @(e_2, i) \geq k$$

  to be part of the specification.
- Obviously, equalities like $\forall i \; @(e_1, i) + @(e_2, i) = k$ may be also part of the extended path RTL specification.
- Then a new and fast algorithm based on a translation to an extended constraint graph is described, too.

- [AnC06''] Stefan Andrei, Albert M.K. Cheng: Faster Verification of RTL-Specified Systems via Decomposition and Constraint Extension, *Real-Time Systems Symposium (RTSS'06)*, December 5-8, 2006, Rio de Janeiro, 10 pages

Divide and Conquer [AnC06"]

- For real-time systems with large specifications, there is a lot of room for improvement in the algorithms used for verification and debugging.
- There is a need of an efficient method to perform verification and debugging of real-time systems specifications using decomposition techniques.
- The idea is to decompose the constraint graph, used in existing approaches, into independent sub-graphs so that it is no longer necessary to analyze the entire specification at once, but rather its individual and smaller components.
- Efficient implementation of this method in the Java-based tool and tested it on several industrial real-time systems.

Future Work

- Identify new subclasses of timing formulae for which the satisfiability problem is decidable:
  - by considering a non-unit scalar integer, e.g.,

    $\pm a * @(X, i) \pm b * @(Y, j) \leq c$
  - by considering more than two variables, e.g.,

    $\pm @(X, i) \pm @(Y, j) \pm @(Z, k) \leq c$

Summary

- Real-time logic
- Counting true instances
- Incremental verification of the real-time systems specifications

Thank you for your attention!

Questions?