Standard BDD operations

- If formulae f_1, f_2 represents sets S_1, S_2, respectively
  then $f_1 \wedge f_2$, $f_1 \vee f_2$ represent $S_1 \cap S_2$, $S_1 \cup S_2$ respectively
- Standard algorithms compute Boolean operation on BDDs
- Abbreviate $(v_1, \ldots, v_n)$ to $\vec{v}$
- If $f(\vec{v})$ represents S
  and $g(\vec{v}, \vec{v}')$ represents $\{(\vec{v}, \vec{v}') \mid R\ \vec{v}\ \vec{v}')\}$
  then $\exists \vec{u}.\ f(\vec{u}) \wedge g(\vec{u}, \vec{v})$ represents $\{\vec{v} \mid \exists \vec{u}.\ \vec{u} \in S \wedge R\ \vec{u}\ \vec{v}\}$
- Can compute BDD of $\exists \vec{u}.\ h(\vec{u}, \vec{v})$ from BDD of $h(\vec{u}, \vec{v})$
  - e.g. BDD of $\exists v_1.\ h(v_1, v_2)$ is BDD of $h(\mathtt{T}, v_2) \vee h(\mathtt{F}, v_2)$
- From BDD of formula $f(v_1, \ldots, v_n)$ can compute $b_1, \ldots, b_n$ such that if $v_1 = b_1, \ldots, v_n = b_n$ then $f(b_1, \ldots, b_n) \Leftrightarrow \mathit{true}$
  - $b_1, \ldots, b_n$ is a satisfying assignment (SAT problem)
  - used for counterexample generation (see later)

Reachable States via BDDs

- Assume $M = (S, S_0, R, L)$ and $S = \mathbb{B}^n$
- Represent R by Boolean formulae $g(\vec{v}, \vec{v'})$
- Iteratively define formula $f_n(\vec{v})$ representing $\mathcal{S}_n$

$$f_0(\vec{v}) \quad = \text{formula representing } S_0$$
$$f_{n+1}(\vec{v}) \quad = f_n(\vec{v}) \vee (\exists \vec{u}.\ f_n(\vec{u}) \wedge g(\vec{u}, \vec{v}))$$

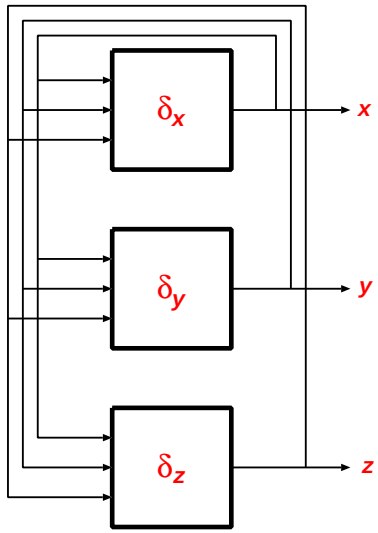
- Let $\mathcal{B}_0$, $\mathcal{B}_R$ be BDDs representing $f_0(\vec{v})$, $g(\vec{v}, \vec{v'})$
- Iteratively compute BDDs $\mathcal{B}_n$ representing f_n

$$\mathcal{B}_{n+1} = \mathcal{B}_n \ \underline{\vee} \ (\underline{\exists \vec{u}.} \ \mathcal{B}_n\underline{[\vec{u}/\vec{v}]} \ \underline{\wedge} \ \mathcal{B}_R\underline{[\vec{u}, \vec{v}/\vec{v}, \vec{v'}]})$$

  - efficient using (blue underlined) standard BDD algorithms (renaming, conjunction, disjunction, quantification)
  - BDD $\mathcal{B}_n$ only contains variables $\vec{v}$: represents $\mathcal{S}_n \subseteq S$

- At each iteration check $\mathcal{B}_{n+1} = \mathcal{B}_n$ efficient using BDDs
  - when $\mathcal{B}_{n+1} = \mathcal{B}_n$ can conclude $\mathcal{B}_n$ represents Reachable M
  - we call this BDD $\mathcal{B}_M$ in a later slide (i.e. $\mathcal{B}_M = \mathcal{B}_n$)

Example BDD optimisation: disjunctive partitioning



Three state transition functions in parallel

$$\delta_x, \delta_y, \delta_z : \mathbb{B} \times \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$$

- Transition relation (asynchronous interleaving semantics):

$$\begin{aligned} &R\ (x, y, z)\ (x', y', z') = \\ &\quad (x' = \delta_x(x, y, z)\ \ \wedge\ \ y' = y\ \ \wedge\ \ z' = z) \vee \\ &\quad (x' = x\ \ \wedge\ \ y' = \delta_y(x, y, z)\ \ \wedge\ \ z' = z) \vee \\ &\quad (x' = x\ \ \wedge\ \ y' = y\ \ \wedge\ \ z' = \delta_z(x, y, z)) \end{aligned}$$

Avoiding building big BDDs

- Transition relation for three transition functions in parallel

$$
\begin{aligned}
&R(x, y, z)\ (x', y', z') = \\
&\quad (x' = \delta_x(x, y, z) \ \wedge \ y' = y \ \wedge \ z' = z) \vee \\
&\quad (x' = x \ \wedge \ y' = \delta_y(x, y, z) \ \wedge \ z' = z) \vee \\
&\quad (x' = x \ \wedge \ y' = y \ \wedge \ z' = \delta_z(x, y, z))
\end{aligned}
$$

- Recall symbolic iteration:

$$f_{n+1}(\vec{v}) \ = f_n(\vec{v}) \vee (\exists \vec{u}.\ f_n(\vec{u}) \wedge g(\vec{u}, \vec{v}))$$

- For this particular R (see next slide):

$$
\begin{aligned}
&f_{n+1}(x, y, z) \\
&\quad = f_n(x, y, z) \vee (\exists \overline{x}\ \overline{y}\ \overline{z}.\ f_n(\overline{x}, \overline{y}, \overline{z}) \wedge R\ (\overline{x}, \overline{y}, \overline{z})\ (x, y, z)) \\
&\quad = f_n(x, y, z) \qquad\qquad\qquad\qquad\qquad\qquad\qquad \vee \\
&\qquad\quad (\exists \overline{x}.\ f_n(\overline{x}, y, z) \ \wedge \ x = \delta_x(\overline{x}, y, z)) \vee \\
&\qquad\quad (\exists \overline{y}.\ f_n(x, \overline{y}, z) \ \wedge \ y = \delta_y(x, \overline{y}, z)) \vee \\
&\qquad\quad (\exists \overline{z}.\ f_n(x, y, \overline{z}) \ \wedge \ z = \delta_z(x, y, \overline{z}))
\end{aligned}
$$

- Don't need to calculate BDD of R!

Disjunctive partitioning – Exercise: understand this

$$\exists \overline{x}\ \overline{y}\ \overline{z}.\ f_n(\overline{x},\overline{y},\overline{z})\ \wedge\ R\ (\overline{x},\overline{y},\overline{z})\ (x,y,z)$$

$$= \exists \overline{x}\ \overline{y}\ \overline{z}.\ f_n(\overline{x},\overline{y},\overline{z})\ \wedge\ ((x=\delta_x(\overline{x},\overline{y},\overline{z})\ \wedge\ y=\overline{y}\ \wedge\ z=\overline{z})\ \vee\ (x=\overline{x}\ \wedge\ y=\delta_y(\overline{x},\overline{y},\overline{z})\ \wedge\ z=\overline{z})\ \vee\ (x=\overline{x}\ \wedge\ y=\overline{y}\ \wedge\ z=\delta_z(\overline{x},\overline{y},\overline{z})))$$

$$= (\exists \overline{x}\ \overline{y}\ \overline{z}.\ f_n(\overline{x},\overline{y},\overline{z})\ \wedge\ x=\delta_x(\overline{x},\overline{y},\overline{z})\ \wedge\ y=\overline{y}\ \wedge\ z=\overline{z})\ \vee\ (\exists \overline{x}\ \overline{y}\ \overline{z}.\ f_n(\overline{x},\overline{y},\overline{z})\ \wedge\ x=\overline{x}\ \wedge\ y=\delta_y(\overline{x},\overline{y},\overline{z})\ \wedge\ z=\overline{z})\ \vee\ (\exists \overline{x}\ \overline{y}\ \overline{z}.\ f_n(\overline{x},\overline{y},\overline{z})\ \wedge\ x=\overline{x}\ \wedge\ y=\overline{y}\ \wedge\ z=\delta_z(\overline{x},\overline{y},\overline{z}))$$

$$= (\exists \overline{x}\ \overline{y}\ \overline{z}.\ f_n(\overline{x},y,z)\ \wedge\ x=\delta_x(\overline{x},y,z)\ \wedge\ y=\overline{y}\ \wedge\ z=\overline{z})\ \vee\ (\exists \overline{x}\ \overline{y}\ \overline{z}.\ f_n(x,\overline{y},z)\ \wedge\ x=\overline{x}\ \wedge\ y=\delta_y(x,\overline{y},z)\ \wedge\ z=\overline{z})\ \vee\ (\exists \overline{x}\ \overline{y}\ \overline{z}.\ f_n(x,y,\overline{z})\ \wedge\ x=\overline{x}\ \wedge\ y=\overline{y}\ \wedge\ z=\delta_z(x,y,\overline{z}))$$

$$= ((\exists \overline{x}.\ f_n(\overline{x},y,z)\wedge x{=}\delta_x(\overline{x},y,z))\ \wedge\ (\exists \overline{y}.\ y{=}\overline{y})\ \wedge\ (\exists \overline{z}.\ z{=}\overline{z}))\ \vee\ ((\exists \overline{x}.\ x{=}\overline{x})\ \wedge\ (\exists \overline{y}.\ f_n(x,\overline{y},z)\wedge y{=}\delta_y(x,\overline{y},z))\ \wedge\ (\exists \overline{z}.\ z{=}\overline{z}))\ \vee\ ((\exists \overline{x}.\ x{=}\overline{x})\ \wedge\ (\exists \overline{y}.\ y{=}\overline{y})\ \wedge\ (\exists \overline{z}.\ f_n(x,y,\overline{z})\wedge z{=}\delta_z(x,y,\overline{z})))$$

$$= (\exists \overline{x}.\ f_n(\overline{x},y,z)\ \wedge\ x=\delta_x(\overline{x},y,z))\ \vee\ (\exists \overline{y}.\ f_n(x,\overline{y},z)\ \wedge\ y=\delta_y(x,\overline{y},z))\ \vee\ (\exists \overline{z}.\ f_n(x,y,\overline{z})\ \wedge\ z=\delta_z(x,y,\overline{z}))$$

Verification and counterexamples

- Typical safety question:
  - is property p true in all reachable states?
  - i.e. check $M \models \mathbf{AG}\, p$
  - i.e. is $\forall s.\ s \in \text{Reachable}\ M \Rightarrow p\ s$
- Check using BDDs
  - compute BDD $\mathcal{B}_M$ of Reachable M
  - compute BDD $\mathcal{B}_p$ of $p(\vec{v})$
  - check if BDD of $\mathcal{B}_M \underline{\Rightarrow} \mathcal{B}_p$ is the single node $\boxed{1}$
- Valid because *true* represented by a unique BDD (canonical property)
- If BDD is not $\boxed{1}$ can get counterexample

Generating counterexamples (general idea)

BDD algorithms can find satisfying assignments (SAT)

- Suppose not all reachable states of model M satisfy p
- i.e. $\exists s \in \text{Reachable } M.\ \neg(p(s))$
- Set of reachable state $\mathcal{S}$ given by: $\mathcal{S} = \bigcup_{n=0}^{\infty} \mathcal{S}_n$
- Iterate to find least n such that $\exists s \in \mathcal{S}_n.\ \neg(p(s))$
- Use SAT to find b_n such that $b_n \in \mathcal{S}_n \wedge \neg(p(b_n))$
- Use SAT to find b_{n-1} such that $b_{n-1} \in \mathcal{S}_{n-1} \wedge R\ b_{n-1}\ b_n$
- Use SAT to find b_{n-2} such that $b_{n-2} \in \mathcal{S}_{n-2} \wedge R\ b_{n-2}\ b_{n-1}$

  $\vdots$
- Iterate to find $b_0, b_1, \ldots, b_{n-1}, b_n$ where $b_i \in \mathcal{S}_i \wedge R\ b_{i-1}\ b_i$
- Then $b_0\ b_1\ \cdots\ b_{n-1}\ b_n$ is a path to a counterexample

Use SAT to find s_{n-1} such that $s_{n-1} \in \mathcal{S}_{n-1} \wedge R\ s_{n-1}\ s_n$

- Suppose states s, s' symbolically represented by $\vec{v}$, $\vec{v'}$
- Suppose BDD $\mathcal{B}_i$ represents $\vec{v} \in \mathcal{S}_i$ $(1 \leq i \leq n)$
- Suppose BDD $\mathcal{B}_R$ represents $R\ \vec{v}\ \vec{v'}$
- Then BDD
  $(\mathcal{B}_{n-1} \mathbin{\underline{\wedge}} \mathcal{B}_R\underline{[\vec{b}_n/\vec{v'}]})$
  represents
  $\vec{v} \in \mathcal{S}_{n-1} \wedge R\ \vec{v}\ \vec{b}_n$
- Use SAT to find a valuation $\vec{b}_{n-1}$ for $\vec{v}$
- Then BDD
  $(\mathcal{B}_{n-1} \mathbin{\underline{\wedge}} \mathcal{B}_R\underline{[\vec{b}_n/\vec{v'}]})\underline{[\vec{b}_{n-1}/\vec{v}]}$
  represents
  $\vec{b}_{n-1} \in \mathcal{S}_{n-1} \wedge R\ \vec{b}_{n-1}\ \vec{b}_n$

Generating counterexamples with BDDs

BDD algorithms can find satisfying assignments (SAT)

- $M = (S, S_0, R, L)$ and $\mathcal{B}_0$, $\mathcal{B}_1$, ..., $\mathcal{B}_M$, $\mathcal{B}_R$, $\mathcal{B}_p$ as earlier
- Suppose $\mathcal{B}_M \Rightarrow \mathcal{B}_p$ is not $\boxed{1}$
- Must exist a state $s \in$ Reachable M such that $\neg(p\ s)$
- Let $\mathcal{B}_{\neg p}$ be the BDD representing $\neg(p\ \vec{v})$
- Iterate to find first n such that $\mathcal{B}_n \wedge \mathcal{B}_{\neg p}$
- Use SAT to find $\vec{b}_n$ such that $(\mathcal{B}_n \wedge \mathcal{B}_{\neg p})[\vec{b}_n/\vec{v}]$
- Use SAT to find $\vec{b}_{n-1}$ such that $(\mathcal{B}_{n-1} \wedge \mathcal{B}_R[\vec{b}_n/\vec{v'}])[\vec{b}_{n-1}/\vec{v}]$
- For $0 < i < n$ find $\vec{b}_{i-1}$ such that $(\mathcal{B}_{i-1} \wedge \mathcal{B}_R[\vec{b}_i/\vec{v'}])[\vec{b}_{i-1}/\vec{v}]$
- $\vec{b}_0$,...,$\vec{b}_i$,...,$\vec{b}_n$ is a counterexample trace
- Sometimes can use partitioning to avoid constructing $\mathcal{B}_R$

==========

CALL FOR PAPERS

2nd ICAPS Workshop on Model Checking and Automated Planning (MOCHAP-15)

http://icaps15.icaps-conference.org/

Jerusalem, Israel, June 7/8, 2015

==========

There has been a lot of work on the exchanges between the two research areas of model checking and automated planning. From a high level perspective, model checking and planning problems are related in the sense that plans (found by a planning system) correspond to error traces (found by a model checker).
For example, finding violations of properties that can be checked on a per-state basis (e.g., mutex properties) in model checking can be achieved by finding goal states in a correspondent planning problem.
Thus, if a plan is found by a planning system, it corresponds to an error trace that a model checker could return. The link can be exploited also in the other way around, using a model checker to search the state space of a planning problem, and stopping the search when a goal state is found. Furthermore, there is a strong connection between hybrid-system falsification and motion planning as
state-of-the-art motion planners are used as the starting point for searching the continuous state spaces of hybrid systems.

The purpose of the workshop is to continue to promote a cross-fertilisation between research on planning and verification, incrementing the synergy between the two areas. This workshop is an ideal venue for
discussing what can be shared in terms of techniques, tools, modelling languages and benchmark problems.

Topics of Interest

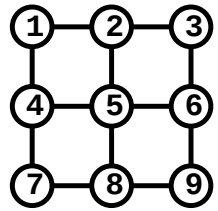
==========

Topics of interest include - but are not limited to - the following topics:

* Planning as model checking

Example (from an exam)

Consider a 3x3 array of 9 switches



Suppose each switch 1,2,...,9 can either be on or off, and that toggling any switch will automatically toggle all its immediate neighbours. For example, toggling switch 5 will also toggle switches 2, 4, 6 and 8, and toggling switch 6 will also toggle switches 3, 5 and 9.

(a) Devise a state space [4 marks] and transition relation [6 marks] to represent the behavior of the array of switches

You are given the problem of getting from an initial state in which even numbered switches are on and odd numbered switches are off, to a final state in which all the switches are off.

(b) Write down predicates on your state space that characterises the initial [2 marks] and final [2 marks] states.

(c) Explain how you might use a model checker to find a sequences of switches to toggle to get from the initial to final state. [6 marks]

You are not expected to actually solve the problem, but only to explain how to represent it in terms of model checking.

Solution

A state is a vector `(v1,v2,v3,v4,v5,v6,v7,v8,v9)`, where `vi` $\in \mathbb{B}$

A transition relation `Trans` is then defined by:

$$
\begin{array}{l}
\texttt{Trans(v1,v2,v3,v4,v5,v6,v7,v8,v9)(v1',v2',v3',v4',v5',v6',v7',v8',v9')} \\
\quad = \texttt{((v1'=}\neg\texttt{v1)}\wedge\texttt{(v2'=}\neg\texttt{v2)}\wedge\texttt{(v3'=v3)}\wedge\texttt{(v4'=}\neg\texttt{v4)}\wedge\texttt{(v5'=v5)}\wedge \\
\qquad \texttt{(v6'=v6)}\wedge\texttt{(v7'=v7)}\wedge\texttt{(v8'=v8)}\wedge\texttt{(v9'=v9))} \qquad \text{(toggle switch 1)} \\
\quad \vee \texttt{((v1'=}\neg\texttt{v1)}\wedge\texttt{(v2'=}\neg\texttt{v2)}\wedge\texttt{(v3'=}\neg\texttt{v3)}\wedge\texttt{(v4'=v4)}\wedge\texttt{(v5'=}\neg\texttt{v5)}\wedge \\
\qquad \texttt{(v6'=v6)}\wedge\texttt{(v7'=v7)}\wedge\texttt{(v8'=v8)}\wedge\texttt{(v9'=v9))} \qquad \text{(toggle switch 2)} \\
\quad \vee \texttt{((v1'=v1)}\wedge\texttt{(v2'=}\neg\texttt{v2)}\wedge\texttt{(v3'=}\neg\texttt{v3)}\wedge\texttt{(v4'=v4)}\wedge\texttt{(v5'=v5)}\wedge \\
\qquad \texttt{(v6'=}\neg\texttt{v6)}\wedge\texttt{(v7'=v7)}\wedge\texttt{(v8'=v8)}\wedge\texttt{(v9'=v9))} \qquad \text{(toggle switch 3)} \\
\quad \vee \texttt{((v1'=}\neg\texttt{v1)}\wedge\texttt{(v2'=v2)}\wedge\texttt{(v3'=v3)}\wedge\texttt{(v4'=}\neg\texttt{v4)}\wedge\texttt{(v5'=}\neg\texttt{v5)}\wedge \\
\qquad \texttt{(v6'=v6)}\wedge\texttt{(v7'=}\neg\texttt{v7)}\wedge\texttt{(v8'=v8)}\wedge\texttt{(v9'=v9))} \qquad \text{(toggle switch 4)} \\
\quad \vee \texttt{((v1'=v1)}\wedge\texttt{(v2'=}\neg\texttt{v2)}\wedge\texttt{(v3'=v3)}\wedge\texttt{(v4'=}\neg\texttt{v4)}\wedge\texttt{(v5'=}\neg\texttt{v5)}\wedge \\
\qquad \texttt{(v6'=}\neg\texttt{v6)}\wedge\texttt{(v7'=v7)}\wedge\texttt{(v8'=}\neg\texttt{v8)}\wedge\texttt{(v9'=v9))} \qquad \text{(toggle switch 5)} \\
\quad \vee \texttt{((v1'=v1)}\wedge\texttt{(v2'=v2)}\wedge\texttt{(v3'=}\neg\texttt{v3)}\wedge\texttt{(v4'=v4)}\wedge\texttt{(v5'=}\neg\texttt{v5)}\wedge \\
\qquad \texttt{(v6'=}\neg\texttt{v6)}\wedge\texttt{(v7'=v7)}\wedge\texttt{(v8'=v8)}\wedge\texttt{(v9'=}\neg\texttt{v9))} \qquad \text{(toggle switch 6)} \\
\quad \vee \texttt{((v1'=v1)}\wedge\texttt{(v2'=v2)}\wedge\texttt{(v3'=v3)}\wedge\texttt{(v4'=}\neg\texttt{v4)}\wedge\texttt{(v5'=v5)}\wedge \\
\qquad \texttt{(v6'=v6)}\wedge\texttt{(v7'=}\neg\texttt{v7)}\wedge\texttt{(v8'=}\neg\texttt{v8)}\wedge\texttt{(v9'=v9))} \qquad \text{(toggle switch 7)} \\
\quad \vee \texttt{((v1'=v1)}\wedge\texttt{(v2'=v2)}\wedge\texttt{(v3'=v3)}\wedge\texttt{(v4'=v4)}\wedge\texttt{(v5'=}\neg\texttt{v5)}\wedge \\
\qquad \texttt{(v6'=v6)}\wedge\texttt{(v7'=}\neg\texttt{v7)}\wedge\texttt{(v8'=}\neg\texttt{v8)}\wedge\texttt{(v9'=}\neg\texttt{v9))} \qquad \text{(toggle switch 8)} \\
\quad \vee \texttt{((v1'=v1)}\wedge\texttt{(v2'=v2)}\wedge\texttt{(v3'=v3)}\wedge\texttt{(v4'=v4)}\wedge\texttt{(v5'=v5)}\wedge \\
\qquad \texttt{(v6'=}\neg\texttt{v6)}\wedge\texttt{(v7'=v7)}\wedge\texttt{(v8'=}\neg\texttt{v8)}\wedge\texttt{(v9'=}\neg\texttt{v9))} \qquad \text{(toggle switch 9)}
\end{array}
$$

Solution (continued)

Predicates `Init`, `Final` characterising the initial and final states, respectively, are defined by:

```
Init(v1,v2,v3,v4,v5,v6,v7,v8,v9) =
 ¬v1 ∧ v2 ∧ ¬v3 ∧ v4 ∧ ¬v5 ∧ v6 ∧ ¬v7 ∧ v8 ∧ ¬v9

Final(v1,v2,v3,v4,v5,v6,v7,v8,v9) =
 ¬v1 ∧ ¬v2 ∧ ¬v3 ∧ ¬v4 ∧ ¬v5 ∧ ¬v6 ∧ ¬v7 ∧ ¬v8 ∧ ¬v9
```

Model checkers can find counter-examples to properties, and sequences of transitions from an initial state to a counter-example state. Thus we could use a model checker to find a trace to a counter-example to the property that

```
¬Final(v1,v2,v3,v4,v5,v6,v7,v8,v9)
```

Properties

- $\forall s \in S_0 . \forall s' . R^*\ s\ s' \Rightarrow p\ s'$ says p true in all reachable states
- Might want to verify other properties
  1. `DeviceEnabled` holds infinitely often along every path
  2. From any state it is possible to get to a state where `Restart` holds
  3. After a three or more consecutive occurrences of `Req` there will eventually be an `Ack`
- Temporal logic can express such properties
- There are several temporal logics in use
  - LTL is good for the first example above
  - CTL is good for the second example
  - PSL is good for the third example
- Model checking:
  - Emerson, Clarke & Sifakis: Turing Award 2008
  - widely used in industry: first hardware, later software

Temporal logic (originally called “tense logic”)

A. N. Prior
1914-1969

Originally devised for investigating: “the relationship between tense and modality attributed to the Megarian philosopher Diodorus Cronus (ca. 340-280 BCE)”.

Mary Prior, his wife, recalls “I remember his waking me one night [in 1953], coming and sitting on my bed, ... and saying he thought one could make a formalised tense logic”.

- Temporal logic: deductive system for reasoning about time
  - temporal formulae for expressing temporal statements
  - deductive system for proving theorems
- Temporal logic model checking
  - uses semantics to check truth of temporal formulae in models
- Temporal logic proof systems also important in CS
  - use pioneered by Amir Pnueli (1996 Turing Award)
  - not considered in this course

Recommended: `http://plato.stanford.edu/entries/prior/`

Temporal logic formulae (statements)

- Many different languages of temporal statements
  - linear time (LTL)
  - branching time (CTL)
  - finite intervals (SEREs)
  - industrial languages (PSL, SVA)
- Prior used linear time, Kripke suggested branching time:

  ... we perhaps should not regard time as a linear series ... there are several possibilities for what the next moment may be like - and for each possible next moment, there are several possibilities for the moment after that. Thus the situation takes the form, not of a linear sequence, but of a 'tree'.

  [Saul Kripke, 1958 (aged 17, still at school)]
- CS issues different from philosophical issues
  - Moshe Vardi: "Branching vs. Linear Time: Final Showdown"

http://www.computer.org/portal/web/awards/Vardi

Moshe Vardi
www.computer.org

"For fundamental and lasting contributions to the development of logic as a unifying foundational framework and a tool for modeling computational systems"

2011 Harry H. Goode Memorial Award Recipient

Linear Temporal Logic (LTL)

- Grammar of *well formed formulae* (wff) ϕ

$$
\begin{array}{lcll}
\phi & ::= & p & \text{(Atomic formula: } p \in AP\text{)} \\
 & | & \neg\phi & \text{(Negation)} \\
 & | & \phi_1 \vee \phi_2 & \text{(Disjunction)} \\
 & | & \mathbf{X}\phi & \text{(successor)} \\
 & | & \mathbf{F}\phi & \text{(sometimes)} \\
 & | & \mathbf{G}\phi & \text{(always)} \\
 & | & [\phi_1\ \mathbf{U}\ \phi_2] & \text{(Until)}
\end{array}
$$

- Details differ from Prior's tense logic – but similar ideas

- Semantics define when ϕ true in model M
  - where $M = (S, S_0, R, L)$ – a Kripke structure
  - notation: $M \models \phi$ means ϕ true in model M
  - model checking algorithms compute this (when decidable)