Communication and Concurrency
Lecture 15

Colin Stirling (cps)

School of Informatics

11th November 2013

Labelled transition systems

- A labelled transition system on a set of actions A is a pair $T = (S, \delta)$, where

Labelled transition systems

- A labelled transition system on a set of actions A is a pair $T = (S, \delta)$, where
- S is a set of states,
- $\delta \subseteq S \times A \times S$ is the transition relation

Labelled transition systems

- A labelled transition system on a set of actions A is a pair $T = (S, \delta)$, where
- S is a set of states,
- $\delta \subseteq S \times A \times S$ is the transition relation
- Restrict to finite transition systems: where A and S (and δ) are finite

Labelled transition systems

- A labelled transition system on a set of actions A is a pair $T = (S, \delta)$, where
- S is a set of states,
- $\delta \subseteq S \times A \times S$ is the transition relation
- Restrict to finite transition systems: where A and S (and δ) are finite
- For process E, we denote by $T_E = (S_E, \delta_E)$ the labelled transition system associated to E, inductively defined as follows:

Labelled transition systems

- A labelled transition system on a set of actions A is a pair $T = (S, \delta)$, where
- S is a set of states,
- $\delta \subseteq S \times A \times S$ is the transition relation
- Restrict to finite transition systems: where A and S (and δ) are finite
- For process E, we denote by $T_E = (S_E, \delta_E)$ the labelled transition system associated to E, inductively defined as follows:

  1. $E \in S_E$, and

Labelled transition systems

- A labelled transition system on a set of actions A is a pair $T = (S, \delta)$, where
- S is a set of states,
- $\delta \subseteq S \times A \times S$ is the transition relation
- Restrict to finite transition systems: where A and S (and δ) are finite
- For process E, we denote by $T_E = (S_E, \delta_E)$ the labelled transition system associated to E, inductively defined as follows:
  1. $E \in S_E$, and

  2. if $F \in S_E$, and $F \xrightarrow{a} G$, then $G \in S_E$ and $(F, a, G) \in \delta_E$.

Model checking CTL$^-$

- Given: a process E, a formula ϕ of CTL$^-$.

Model checking CTL$^-$

- Given: a process E, a formula ϕ of CTL$^-$.
- Decide: does E satisfies ϕ ?

Model checking CTL$^-$

- Given: a process E, a formula ϕ of CTL$^-$.
- Decide: does E satisfies ϕ ?
- For convenience we add negation to CTL$^-$

Model checking CTL$^-$

- Given: a process E, a formula ϕ of CTL$^-$.
- Decide: does E satisfies ϕ ?
- For convenience we add negation to CTL$^-$
- For formula ψ, $[\![\psi]\!]$ is the set of states of T_E satisfying ψ.

Model checking CTL$^-$

- Given: a process E, a formula ϕ of CTL$^-$.
- Decide: does E satisfies ϕ ?
- For convenience we add negation to CTL$^-$
- For formula ψ, $[\![\psi]\!]$ is the set of states of T_E satisfying ψ.
- Sketch of the algorithm:

Model checking CTL$^-$

- Given: a process E, a formula ϕ of CTL$^-$.
- Decide: does E satisfies ϕ ?
- For convenience we add negation to CTL$^-$
- For formula ψ, $[\![\psi]\!]$ is the set of states of T_E satisfying ψ.
- Sketch of the algorithm:
  1. Compute the subformulas of ϕ

Model checking CTL$^-$

- Given: a process E, a formula ϕ of CTL$^-$.
- Decide: does E satisfies ϕ ?
- For convenience we add negation to CTL$^-$
- For formula ψ, $[\![\psi]\!]$ is the set of states of T_E satisfying ψ.
- Sketch of the algorithm:
  1. Compute the subformulas of ϕ
  2. Compute $[\![\psi]\!]$ for each subformula ψ of ϕ, starting with the smallest subformulas and then with larger and larger subformulas

Model checking CTL$^-$

- Given: a process E, a formula ϕ of CTL$^-$.
- Decide: does E satisfies ϕ ?
- For convenience we add negation to CTL$^-$
- For formula ψ, $[\![\psi]\!]$ is the set of states of T_E satisfying ψ.
- Sketch of the algorithm:
  1. Compute the subformulas of ϕ
  2. Compute $[\![\psi]\!]$ for each subformula ψ of ϕ, starting with the smallest subformulas and then with larger and larger subformulas
  3. Answer: "E satisfies ϕ" iff $E \in [\![\phi]\!]$

Computing $[\![\psi]\!]$: the easy cases

- Because of the equivalences

$$\begin{aligned} [K]\psi &\equiv \neg\langle K\rangle\neg\psi \\ \mathtt{AG}\ \psi &\equiv \neg\mathtt{EF}\ \neg\psi \\ \mathtt{AF}\ \psi &\equiv \neg\mathtt{EG}\ \neg\psi \end{aligned}$$

we can assume that ϕ does not contain $[K]$, AG or AF operators

$$\begin{aligned} [\![\mathtt{tt}]\!] &= S_E \\ [\![\mathtt{ff}]\!] &= \emptyset \\ [\![\psi_1 \wedge \psi_2]\!] &= [\![\psi_1]\!] \cap [\![\psi_2]\!] \\ [\![\psi_1 \vee \psi_2]\!] &= [\![\psi_1]\!] \cup [\![\psi_2]\!] \\ [\![\neg\psi]\!] &= S_E \setminus [\![\psi]\!] \\ [\![\langle K\rangle\psi_1]\!] &= pre_K([\![\psi_1]\!]) \end{aligned}$$

Computing $[\![\psi]\!]$: the easy cases

- Because of the equivalences

$$
\begin{array}{rcl}
[K]\psi & \equiv & \neg\langle K\rangle\neg\psi \\
\mathtt{AG}\ \psi & \equiv & \neg\mathtt{EF}\ \neg\psi \\
\mathtt{AF}\ \psi & \equiv & \neg\mathtt{EG}\ \neg\psi
\end{array}
$$

we can assume that ϕ does not contain $[K]$, $\mathtt{AG}$ or $\mathtt{AF}$ operators

$$
\begin{array}{rcl}
[\![\mathtt{tt}]\!] & = & S_E \\
[\![\mathtt{ff}]\!] & = & \emptyset \\
[\![\psi_1 \wedge \psi_2]\!] & = & [\![\psi_1]\!] \cap [\![\psi_2]\!] \\
[\![\psi_1 \vee \psi_2]\!] & = & [\![\psi_1]\!] \cup [\![\psi_2]\!] \\
[\![\neg\psi]\!] & = & S_E \setminus [\![\psi]\!] \\
[\![\langle K\rangle\psi_1]\!] & = & pre_K([\![\psi_1]\!])
\end{array}
$$

- $pre_K([\![\psi_1]\!]) \stackrel{\text{def}}{=}$ states from which some state in $[\![\psi_1]\!]$ can be reached through some action in K.

Computing $[\![\psi]\!]$: the easy cases

- Because of the equivalences

$$
\begin{array}{rcl}
[K]\psi & \equiv & \neg\langle K\rangle\neg\psi \\
\mathtt{AG}\ \psi & \equiv & \neg\mathtt{EF}\ \neg\psi \\
\mathtt{AF}\ \psi & \equiv & \neg\mathtt{EG}\ \neg\psi
\end{array}
$$

we can assume that ϕ does not contain $[K]$, $\mathtt{AG}$ or $\mathtt{AF}$ operators

$$
\begin{array}{rcl}
[\![\mathtt{tt}]\!] & = & S_E \\
[\![\mathtt{ff}]\!] & = & \emptyset \\
[\![\psi_1 \wedge \psi_2]\!] & = & [\![\psi_1]\!] \cap [\![\psi_2]\!] \\
[\![\psi_1 \vee \psi_2]\!] & = & [\![\psi_1]\!] \cup [\![\psi_2]\!] \\
[\![\neg\psi]\!] & = & S_E \setminus [\![\psi]\!] \\
[\![\langle K\rangle\psi_1]\!] & = & pre_K([\![\psi_1]\!])
\end{array}
$$

- $pre_K([\![\psi_1]\!]) \stackrel{\text{def}}{=}$ states from which some state in $[\![\psi_1]\!]$ can be reached through some action in K.
- Complexity: $O(|S_E| + |\delta_E|)$

Computing $[\![\mathrm{EF}\ \psi_1]\!]$

Let $pre(\ )$ denote $pre_A(\ )$

> Input: T_E, $[\![\psi_1]\!]$
> Output $[\![\mathrm{EF}\ \psi_1]\!]$
>
> Initialize $C := [\![\psi_1]\!]$;
>
> Iterate $C := C \cup pre(C)$
> until a fixpoint is reached;
>
> return C

Complexity: $O(|S_E| \cdot (|S_E| + |\delta_E|)$

Better algorithm: explore each state only once using depth-first or breadth-first search. (Complexity: $O(|S_E| + |\delta_E|)$)

Computing $[\![\mathrm{EG}\ \psi_1]\!]$

> Input: T_E, $[\![\psi_1]\!]$
> Output: $[\![\mathrm{EG}\ \psi_1]\!]$
>
> Compute $D :=$ states of S_E without successors;
>
> Initialize $C := S_E$;
>
> Iterate $C := [\![\psi_1]\!] \cap (pre(C) \cup D)$
>
> until a fixpoint is reached;
>
> return C

Complexity: $O(|S_E| \cdot (|S_E| + |\delta_E|)$

Computing $[\![\mathrm{EG}\ \psi_1]\!]$ II

An algorithm with $O(|S_E| + |\delta_E|)$ complexity:

- Compute $D' :=$ states of S_E without successors in $[\![\psi_1]\!]$
- Compute the labelled transition system $T'_E = ([\![\psi_1]\!], \delta_E \cap ([\![\psi_1]\!] \times A \times [\![\psi_1]\!])$
- Compute the set of states C that belong to some strongly connected component of T'_E.
- Using the algorithm for $[\![\mathrm{EF}\ \psi_1]\!]$ case, compute the states from which some state in $C \cup D'$ can be reached (using transitions of T'_E only).

Complexity of the complete model-checking algorithm

- A formula ϕ has at most $|\phi|$ subformulas (where $|\phi|$ is the length of ϕ).

Complexity of the complete model-checking algorithm

- A formula ϕ has at most $|\phi|$ subformulas (where $|\phi|$ is the length of ϕ).
- So the algorithms for the easy cases, for $\mathrm{EF}\ \psi$, and for $\mathrm{EG}\ \psi$ have to be executed altogether at most $|\phi|$ times

Complexity of the complete model-checking algorithm

- A formula ϕ has at most $|\phi|$ subformulas (where $|\phi|$ is the length of ϕ).
- So the algorithms for the easy cases, for $\mathrm{EF}\ \psi$, and for $\mathrm{EG}\ \psi$ have to be executed altogether at most $|\phi|$ times
- Each execution of one of the algorithms takes at most $O(|S_E| + |\delta_E|)$ time (using the fast algorithms).

Complexity of the complete model-checking algorithm

- A formula ϕ has at most $|\phi|$ subformulas (where $|\phi|$ is the length of ϕ).
- So the algorithms for the easy cases, for $\mathtt{EF}\ \psi$, and for $\mathtt{EG}\ \psi$ have to be executed altogether at most $|\phi|$ times
- Each execution of one of the algorithms takes at most $O(|S_E| + |\delta_E|)$ time (using the fast algorithms).
- So the overall complexity is

$$O(|\phi| \cdot (|S_E| + |\delta_E|))$$

Fixpoint view of the algorithms

- The following equivalences hold:

$$\begin{aligned} \mathtt{EF}\ \phi &\equiv \phi \vee \langle - \rangle \mathtt{EF}\ \phi \\ \mathtt{EG}\ \phi &\equiv \phi \wedge (\langle - \rangle \mathtt{EG}\ \phi \vee [-]\mathtt{ff}) \end{aligned}$$

Fixpoint view of the algorithms

- The following equivalences hold:

$$\begin{aligned} \mathtt{EF}\ \phi &\equiv \phi \vee \langle - \rangle \mathtt{EF}\ \phi \\ \mathtt{EG}\ \phi &\equiv \phi \wedge (\langle - \rangle \mathtt{EG}\ \phi \vee [-]\mathtt{ff}) \end{aligned}$$

- So we have

$$\begin{aligned} [\![\mathtt{EF}\ \phi]\!] &= [\![\phi]\!] \cup pre([\![\mathtt{EF}\ \phi]\!]) \\ [\![\mathtt{EG}\ \phi]\!] &= [\![\phi]\!] \cap (pre([\![\mathtt{EG}\ \phi]\!]) \cup [\![[-]\mathtt{ff}]\!]) \end{aligned}$$

Fixpoint view of the algorithms

- The following equivalences hold:

$$\begin{aligned} \mathtt{EF}\ \phi &\equiv \phi \vee \langle - \rangle \mathtt{EF}\ \phi \\ \mathtt{EG}\ \phi &\equiv \phi \wedge (\langle - \rangle \mathtt{EG}\ \phi \vee [-]\mathtt{ff}) \end{aligned}$$

- So we have

$$\begin{aligned} [\![\mathtt{EF}\ \phi]\!] &= [\![\phi]\!] \cup pre([\![\mathtt{EF}\ \phi]\!]) \\ [\![\mathtt{EG}\ \phi]\!] &= [\![\phi]\!] \cap (pre([\![\mathtt{EG}\ \phi]\!]) \cup [\![[-]\mathtt{ff}]\!]) \end{aligned}$$

- and so $[\![\mathtt{EF}\ \phi]\!]$ and $[\![\mathtt{EG}\ \phi]\!]$ are solutions of the equations

$$\begin{aligned} X &= [\![\phi]\!] \cup pre(X) &&\stackrel{\mathrm{def}}{=} ef(X) \\ X &= [\![\phi]\!] \cap (pre(X) \cup [\![[-]\mathtt{ff}]\!] &&\stackrel{\mathrm{def}}{=} eg(X) \end{aligned}$$

Fixpoint view of the algorithms

- The following equivalences hold:

$$\begin{array}{lcl} \mathtt{EF}\ \phi & \equiv & \phi \vee \langle - \rangle \mathtt{EF}\ \phi \\ \mathtt{EG}\ \phi & \equiv & \phi \wedge (\langle - \rangle \mathtt{EG}\ \phi \vee [-]\mathtt{ff}) \end{array}$$

- So we have

$$\begin{array}{lcl} [\![\mathtt{EF}\ \phi]\!] & = & [\![\phi]\!] \cup pre([\![\mathtt{EF}\ \phi]\!]) \\ [\![\mathtt{EG}\ \phi]\!] & = & [\![\phi]\!] \cap (pre([\![\mathtt{EG}\ \phi]\!]) \cup [\![[-]\mathtt{ff}]\!]) \end{array}$$

- and so $[\![\mathtt{EF}\ \phi]\!]$ and $[\![\mathtt{EG}\ \phi]\!]$ are solutions of the equations

$$\begin{array}{lcll} X & = & [\![\phi]\!] \cup pre(X) & \stackrel{\text{def}}{=} ef(X) \\ X & = & [\![\phi]\!] \cap (pre(X) \cup [\![[-]\mathtt{ff}]\!] & \stackrel{\text{def}}{=} eg(X) \end{array}$$

- These solutions are fixpoints of the mappings ef and eg.

Which solutions?

Proposition: $[\![\mathtt{EF}\ \phi]\!]$ is the smallest solution (least fixpoint) of $X = ef(X)$.

Which solutions?

Proposition: $[\![\mathtt{EF}\ \phi]\!]$ is the smallest solution (least fixpoint) of $X = ef(X)$.

Proof: Notice that $[\![\mathtt{EF}\ \phi]\!] = \bigcup_{i \geq 0} pre^i([\![\phi]\!])$, where $pre^0([\![\phi]\!]) \stackrel{\text{def}}{=} [\![\phi]\!]$.
Let X_0 be an arbitrary solution.
We prove $pre^i([\![\phi]\!]) \subseteq X_0$ for every $i \geq 0$ by induction on i.

Which solutions?

Proposition: $[\![\mathtt{EF}\ \phi]\!]$ is the smallest solution (least fixpoint) of $X = ef(X)$.

Proof: Notice that $[\![\mathtt{EF}\ \phi]\!] = \bigcup_{i \geq 0} pre^i([\![\phi]\!])$, where $pre^0([\![\phi]\!]) \stackrel{\text{def}}{=} [\![\phi]\!]$.
Let X_0 be an arbitrary solution.
We prove $pre^i([\![\phi]\!]) \subseteq X_0$ for every $i \geq 0$ by induction on i.
Base: $i = 0$. Obvious from $X_0 = [\![\phi]\!] \cup$ "something".
Step: Assume $pre^i([\![\phi]\!]) \subseteq X_0$. Then:

$$\begin{array}{lll} & pre^{i+1}([\![\phi]\!]) & \\ = & pre(pre^i([\![\phi]\!])) & \text{(definition of } pre\text{)} \\ \subseteq & pre(X_0) & \text{(induction hypothesis)} \\ \subseteq & X_0 & (X_0 = pre(X_0) \cup \text{"something"}) \end{array}$$

Which solutions?

Proposition: $[\![\mathrm{EF}\ \phi]\!]$ is the smallest solution (least fixpoint) of $X = ef(X)$.

Proof: Notice that $[\![\mathrm{EF}\ \phi]\!] = \bigcup_{i\geq 0} pre^i([\![\phi]\!])$, where $pre^0([\![\phi]\!]) \stackrel{\text{def}}{=} [\![\phi]\!]$.
Let X_0 be an arbitrary solution.
We prove $pre^i([\![\phi]\!]) \subseteq X_0$ for every $i \geq 0$ by induction on i.
Base: $i = 0$. Obvious from $X_0 = [\![\phi]\!] \cup$ "something".
Step: Assume $pre^i([\![\phi]\!]) \subseteq X_0$. Then:

$$\begin{array}{lll} & pre^{i+1}([\![\phi]\!]) & \\ = & pre(pre^i([\![\phi]\!])) & \text{(definition of } pre\text{)} \\ \subseteq & pre(X_0) & \text{(induction hypothesis)} \\ \subseteq & X_0 & (X_0 = pre(X_0) \cup \text{"something"}) \end{array}$$

Proposition: $[\![\mathrm{EG}\ \phi]\!]$ is the largest solution (greatest fixpoint) of $X = eg(X)$.
Proof: Exercise

Fixpoint algorithms

- The mappings $ef(X)$ and $eg(X)$ are monotonic, i.e., if $X \subseteq Y$, then $ef(X) \subseteq ef(Y)$ and $eg(X) \subseteq eg(Y)$

Fixpoint algorithms

- The mappings $ef(X)$ and $eg(X)$ are monotonic, i.e., if $X \subseteq Y$, then $ef(X) \subseteq ef(Y)$ and $eg(X) \subseteq eg(Y)$
- Given a finite set S and a monotonic mapping $m\colon 2^S \to 2^S$,

Fixpoint algorithms

- The mappings $ef(X)$ and $eg(X)$ are monotonic, i.e., if $X \subseteq Y$, then $ef(X) \subseteq ef(Y)$ and $eg(X) \subseteq eg(Y)$
- Given a finite set S and a monotonic mapping $m\colon 2^S \to 2^S$,
- the least fixpoint of m exists, is unique, and can be calculated by iteratively computing $\emptyset, m(\emptyset), m^2(\emptyset), \ldots$ until $m^i(\emptyset) = m^{i+1}(\emptyset)$. The least fixpoint is $m^i(\emptyset)$;

Fixpoint algorithms

- The mappings $ef(X)$ and $eg(X)$ are monotonic, i.e., if $X \subseteq Y$, then $ef(X) \subseteq ef(Y)$ and $eg(X) \subseteq eg(Y)$
- Given a finite set S and a monotonic mapping $m\colon 2^S \to 2^S$,
- the least fixpoint of m exists, is unique, and can be calculated by iteratively computing $\emptyset, m(\emptyset), m^2(\emptyset), \ldots$ until $m^i(\emptyset) = m^{i+1}(\emptyset)$. The least fixpoint is $m^i(\emptyset)$;
- the greatest fixpoint of m exists, is unique, and can be calculated by iteratively computing $S, m(S), m^2(S), \ldots$ until $m^i(S) = m^{i+1}(S)$. The greatest fixpoint is $m^i(S)$.

Applications

- Fixpoint theory allows to easily derive algorithms for other temporal operators.

$$E_0 \models \phi \;\mathrm{EU}\; \psi \quad \text{iff} \quad \begin{array}{l}\text{for some run } E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \cdots, \\ \text{for some } i \geq 0,\ E_i \models \psi, \text{ and} \\ \text{for all } j < i,\ E_j \models \phi\end{array}$$

Applications

- Fixpoint theory allows to easily derive algorithms for other temporal operators.

$$E_0 \models \phi \;\mathrm{EU}\; \psi \quad \text{iff} \quad \begin{array}{l}\text{for some run } E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \cdots, \\ \text{for some } i \geq 0,\ E_i \models \psi, \text{ and} \\ \text{for all } j < i,\ E_j \models \phi\end{array}$$

- Equivalence: $\phi \;\mathrm{EU}\; \psi \;\equiv\; \psi \vee (\phi \wedge \langle - \rangle \phi \;\mathrm{EU}\; \psi)$

Applications

- Fixpoint theory allows to easily derive algorithms for other temporal operators.

$$E_0 \models \phi \;\mathrm{EU}\; \psi \quad \text{iff} \quad \begin{array}{l}\text{for some run } E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \cdots, \\ \text{for some } i \geq 0,\ E_i \models \psi, \text{ and} \\ \text{for all } j < i,\ E_j \models \phi\end{array}$$

- Equivalence: $\phi \;\mathrm{EU}\; \psi \;\equiv\; \psi \vee (\phi \wedge \langle - \rangle \phi \;\mathrm{EU}\; \psi)$
- So: $[\![\phi \;\mathrm{EU}\; \psi]\!] \;=\; [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathit{pre}([\![\phi \;\mathrm{EU}\; \psi]\!]))$

Applications

- Fixpoint theory allows to easily derive algorithms for other temporal operators.

$$E_0 \models \phi \ \mathtt{EU}\ \psi \quad \text{iff} \quad \begin{array}{l} \text{for some run } E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \cdots, \\ \text{for some } i \geq 0,\ E_i \models \psi, \text{ and} \\ \text{for all } j < i,\ E_j \models \phi \end{array}$$

- Equivalence: $\phi \ \mathtt{EU}\ \psi \ \equiv \ \psi \vee (\phi \wedge \langle - \rangle \phi \ \mathtt{EU}\ \psi)$
- So: $[\![\phi \ \mathtt{EU}\ \psi]\!] \ = \ [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathit{pre}([\![\phi \ \mathtt{EU}\ \psi]\!]))$
- $[\![\phi \ \mathtt{EU}\ \psi]\!]$ is solution of the equation

$$X \ = \ [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathit{pre}(X)) \ \stackrel{\text{def}}{=} \ \mathit{eu}(X)$$

Applications

- Fixpoint theory allows to easily derive algorithms for other temporal operators.

$$E_0 \models \phi \ \mathtt{EU}\ \psi \quad \text{iff} \quad \begin{array}{l} \text{for some run } E_0 \xrightarrow{a_1} E_1 \xrightarrow{a_2} \cdots, \\ \text{for some } i \geq 0,\ E_i \models \psi, \text{ and} \\ \text{for all } j < i,\ E_j \models \phi \end{array}$$

- Equivalence: $\phi \ \mathtt{EU}\ \psi \ \equiv \ \psi \vee (\phi \wedge \langle - \rangle \phi \ \mathtt{EU}\ \psi)$
- So: $[\![\phi \ \mathtt{EU}\ \psi]\!] \ = \ [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathit{pre}([\![\phi \ \mathtt{EU}\ \psi]\!]))$
- $[\![\phi \ \mathtt{EU}\ \psi]\!]$ is solution of the equation

$$X \ = \ [\![\psi]\!] \cup ([\![\phi]\!] \cap \mathit{pre}(X)) \ \stackrel{\text{def}}{=} \ \mathit{eu}(X)$$

- It is the smallest solution, and so, since *eu* is monotonic, we can compute $[\![\phi \ \mathtt{EU}\ \psi]\!]$ as the stabilizing point of $\emptyset, \mathit{eu}(\emptyset), \mathit{eu}^2(\emptyset), \ldots$.

The logic of the Workbench

- In order to encode CTL$^-$ in the Workbench's logic, we write

```
prop AG(P) = max(Z.P & [-]Z);
prop EF(P) = min(X.P | <->X);
prop AF(P) = min(X.P | (<->T & [-]X));
prop EG(P) = max(X.P &([-]F | <->X));
```

The logic of the Workbench

- In order to encode CTL$^-$ in the Workbench's logic, we write

```
prop AG(P) = max(Z.P & [-]Z);
prop EF(P) = min(X.P | <->X);
prop AF(P) = min(X.P | (<->T & [-]X));
prop EG(P) = max(X.P &([-]F | <->X));
```

- These definitions correspond to recursive equations.

The logic of the Workbench

- In order to encode CTL$^-$ in the Workbench's logic, we write
  ```
  prop AG(P) = max(Z.P & [-]Z);
  prop EF(P) = min(X.P | <->X);
  prop AF(P) = min(X.P | (<->T & [-]X));
  prop EG(P) = max(X.P &([-]F | <->X));
  ```
- These definitions correspond to recursive equations.
- E.g., the definition of `EF` ϕ states that $[\![\texttt{EF}\ \phi]\!]$ is the smallest solution (`min`) of the equation

$$X = [\![\phi]\!] \vee pre(X)$$

The logic of the Workbench

- In order to encode CTL$^-$ in the Workbench's logic, we write
  ```
  prop AG(P) = max(Z.P & [-]Z);
  prop EF(P) = min(X.P | <->X);
  prop AF(P) = min(X.P | (<->T & [-]X));
  prop EG(P) = max(X.P &([-]F | <->X));
  ```
- These definitions correspond to recursive equations.
- E.g., the definition of `EF` ϕ states that $[\![\texttt{EF}\ \phi]\!]$ is the smallest solution (`min`) of the equation

$$X = [\![\phi]\!] \vee pre(X)$$

- In other words, in the Workbench a property is defined through a (possibly recursive) equation.