Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

# Logics of Programs

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

## Logics of Programs

- Are modal logics that allow to describe properties of transition systems
- Examples:
  - HennesyMilner Logic
  - Propositional Dynamic Logics
  - Modal (Propositional) Mu-calculus
- Perfectly suited for describing transition systems: they can tell apart transition systems modulo bisimulation



Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

## HennessyMilner Logic

*HM Logic aka (multi) modal logic Ki*

- Syntax:

$\Phi$ := Final | P *(atomic propositions)*
[a]$\Phi$ | <a>$\Phi$ *(modal operators)*
$\neg\Phi$ | $\Phi_1 \wedge \Phi_2$ | $\Phi_1 \vee \Phi_2$ | true | false *(closed under booleans)*

- Propositions are used to denote final states and other TS atomic properties
- <a>$\Phi$ means there exists an a-transition that leads to a state where $\Phi$ holds; i.e., expresses the capability of executing action a bringing about $\Phi$
- [a]$\Phi$ means that all a-transitions lead to states where $\Phi$ holds; i.e., express that executing action a brings about $\Phi$



Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
SAPIENZA Università di Roma

## HennessyMilner Logic

- Semantics: assigns meaning to the formulas.
- Given a TS T = < A, S, $S^0$, $\delta$, F>, a state s $\in$ S, and a formula $\Phi$, we define (by structural induction) the "truth relation"

$$T,s \models \Phi$$

  - T,s $\models$ Final if s $\in$ F (similarly T,s $\models$ P if s $\in$ P);
  - T,s $\models$ [a] $\Phi$ if **for all** s' such that s $\rightarrow_a$ s' we have T,s' $\models$ $\Phi$;
  - T,s $\models$ $\langle a \rangle\Phi$ if **exists** s' such that s $\rightarrow_a$ s' and T,s' $\models$ $\Phi$;
  - T,s $\models$ $\neg\Phi$ if it is not the case that T,s $\models$ $\Phi$;
  - T,s $\models$ $\Phi_1 \vee \Phi_2$ if T,s $\models$ $\Phi_1$ or T,s $\models$ $\Phi_2$ ;
  - T,s $\models$ $\Phi_1 \wedge \Phi_2$ if T,s $\models$ $\Phi_1$ and T,s $\models$ $\Phi_2$ ;
  - T,s $\models$ true always;
  - T,s $\models$ false never.

## HennessyMilner Logic

- Another way to give the same semantics to formulas: formulas extension in a transition system assigns meaning to the formulas.
- Given a TS $T = \langle A, S, S^0, \delta, F\rangle$ "the extension of a formula $\Phi$ in T", denote by $(\Phi)^T$, is defined as follows:
  - $(Final)^T$ = $F$ (similarly $P^T = \{s \mid s \in P\}$);
  - $([a]\,\Phi)^T$ = $\{s \mid \forall s'.\ s \rightarrow_a s' \text{ implies } s' \in (\Phi)^T\}$;
  - $(\langle a\rangle\Phi)^T$ = $\{s \mid \exists s'.\ s \rightarrow_a s' \text{ and } s' \in (\Phi)^T\}$;
  - $(\neg\Phi)^T$ = $S - (\Phi)^T$ ;
  - $(\Phi_1 \vee \Phi_2)^T$ = $(\Phi_1)^T \cup (\Phi_2)^T$ ;
  - $(\Phi_1 \wedge \Phi_2)^T$ = $(\Phi_1)^T \cap (\Phi_2)^T$;
  - $(true)^T$ = $S$;
  - $(false)^T$ = $\emptyset$.
- Note: $T,s \models \Phi$ now written as $s \in (\Phi)^T$



## Model Checking

- Given a TS T, one of its states s, and a formula $\Phi$ verify whether the formula holds in s. Formally:

  $T,s \models \Phi$ or $s \in (\Phi)^T$

- Examples (TS is our vending machine):
  - $S_0 \models$ Final
  - $S_0 \models$ <10c>true — *capability of performing action 10c*
  - $S_2 \models$ [big]false — *inability of performing action big*
  - $S_0 \models$ [10c][big]false — *after 10c cannot execute big*
- Model checking variant (aka "query answering"):
  - Given a TS T … — *the database*
  - … compute the extension of $\Phi$ — *the query*

  Formally: compute the set $(\Phi)^T$ which is equal to $\{s \mid T,s \models \Phi\}$



## Satisfiability

- Satisfiability: given a formula $\Phi$ verify whether there exists a (finite/infinite) TS T and a state of T such that the formula holds in s.

  SAT: check the existance of T,s such that $T,s \models \Phi$

- Validity: given a formula $\Phi$ verify whether in every (finite/infinite) TS T and in every state of T the formula holds in s.

  VAL: check the non existance of T,s such that $T,s \models \neg\Phi$

  Note: VAL = non SAT

Examples: check the satifiability / validity of the following formulas:
- <10p><small><collect$_s$>Final
- Final →



## HennessyMilner Logic and Bisimulation

- Consider two TS, $T = (A,S,s_0,\delta, F)$ and $T' = (A,S',t_0,\delta', F')$.
- Let L be the language formed by all HennessyMilner Logic formulas.
- We define:
  - $\sim_L = \{(s,t) \mid$ for all $\Phi$ of L we have $T,s \models \Phi$ iff $T,s \models \Phi\}$
  - $\sim = \{(s,t) \mid$ exists a bisimulation R s.t., $R(s,t)\}$
- **Theorem**: $s \sim_L t$ iff $s \sim t$
- Proof: we show that
  - $s \sim t$ implies $s \sim_L t$ by structural induction on formulas of L.
  - $s \sim_L t$ implies $s \sim t$ by coinduction showing that $s \sim_L t$ is a bisimulation.

*This theorem says that HennessyMilner Logic has exactly the same distinguishing power of bisimulation.*
*So L is the right logic to predicate on transition systems.*

## *Examples*

- Usefull abbreviation (let actions $A = \{a_{1,\ldots,} a_n\}$):
  - $\langle any \rangle\ \Phi$ stands for $\langle a_1\rangle\Phi \vee \cdots \vee \langle a_n\rangle\Phi$
  - $[any]\ \Phi$ stands for $[a_1]\Phi \wedge \cdots \wedge [a_n]\Phi$
  - $\langle any - a_1\rangle\ \Phi$ stands for $\langle a_2\rangle\Phi \vee \cdots \vee \langle a_v\rangle\Phi$
  - $[any - a_1]\ \Phi$ stands for $[a_2]\Phi \wedge \cdots \wedge [a_v]\Phi$

- Examples:
  - $\langle a\rangle true$ *cabability of performing action a*
  - $[a]false$ *inability of performing action a*
  - $\neg Final \wedge \langle any\rangle true \wedge [any\text{-}a]false$ *necessity/inevitability of performing action a (i.e., action a is the only action possible)*
  - $\neg Final \wedge [any]false$ *deadlock!*



## *Propositional Dynamic Logic*

- $\Phi := P \mid$ *(atomic propositions)*
  $\neg\ \Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid$ *(closed under boolean operators)*
  $[r]\Phi \mid \langle r\rangle\Phi$ *(modal operators)*

  $r := a \mid r_1 + r_2 \mid r_1;r_2 \mid r^* \mid P?$ *(complex actions as regular expressions)*

- Essentially add the capability of expressing partial correctness assertions via formulas of the form
  - $\Phi_1 \rightarrow [r]\Phi_2$ *under the conditions $\Phi_1$ all possible executions of r that terminate reach a state of the TS where $\Phi_2$ holds*

- Also add the ability of asserting that a property holds in all nodes of the transition system
  - $[(a_1 + \cdots + a_v)^*]\Phi$ *in every reachable state of the TS $\Phi$ holds*

- Useful abbereviations:
  - any stands for $(a_1 + \cdots + a_v)$ *Note that + can be expressed also in HM Logic*
  - u stands for any* *This is the so called master/universal modality*



## *Modal Mu-Calculus*

- $\Phi := P \mid$ *(atomic propositions)*
  $\neg\ \Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid$ *(closed under boolean operators)*
  $[r]\Phi \mid \langle r\rangle\Phi$ *(modal operators)*

  $\mu\ X.\Phi(X) \mid \nu\ X.\Phi(X)$ *(fixpoint operators)*

- It is the most expressive logic of the family of logics of programs.
- It subsumes
  - PDL (modalities involving complex actions are translated into fomulas involving fixpoints)
  - LTL (linear time temporal logic),
  - CTS, CTS* (branching time temporal logics)

- Examples:
- $[any^*]\Phi$ can be expressed as $\nu\ X.\ \Phi \wedge [any]X$

- $\mu\ X.\ \Phi \vee [any]X$ *along all runs eventually $\Phi$*
- $\mu\ X.\ \Phi \vee \langle any\rangle X$ *along some run eventually $\Phi$*
- $\nu\ X.\ [a](\mu\ Y.\ \langle any\rangle true \wedge [any\text{-}b]Y) \wedge X$ *every run that that contains a contains later b*



## *Examples of Modal Mu-Calculus*

- Examples (TS is our vending machine):
  - $S_0 \models Final$
  - $S_0 \models \langle 10c\rangle true$ *capability of performing action 10c*
  - $S_2 \models [big]false$ *inability of performing action big*
  - $S_0 \models [10c][big]false$ *after 10c cannot execute big*
  - $S_i \models \mu\ X.\ Final \vee [any]\ X$ *eventually a final state is reached*
  - $S_0 \models \nu\ Z.\ (\mu\ X.\ Final \vee [any]\ X) \wedge [any]\ Z$ *or equivalently*
    $S_0 \models [any^*](\mu\ X.\ Final \vee [any]\ X)$ *from everywhere eventually final*

# Model Checking/Satisfiability

- Model checking is polynomial in the size of the TS for
  - HennessyMilner Logic
  - PDL
  - Modal Mu-Calculus
- Also model checking is wrt the formula
  - Polynomial for HennessyMiner Logic
  - Polynomial for PDL
  - Polynomial for Modal Mu-Calculus with bounded alternation of fixpoints and NP∩coNP in general
- Satisfiability is decidable for the three logics, and the complexity (in the size of the formula) is as follows:
  - HennessyMilner Logic: PSPACE-complete
  - PDL: EXPTIME-complete
  - Modal Mu-Calculus: EXPTIME-complete



# AI Planning as Model Checking

- **Build the TS of the domain:**
  - Consider the set of states formed all possible truth value of the propositions (this works only for propositional setting).
  - Use Pre's and Post of actions for determining the transitions

  *Note: the TS is exponential in the size od the description.*
- **Write the goal in a logic of program**
  - typically a single least fixpoint formula of Mu-Calculus (compute ***reachable*** states intersection states where goal true)
- **Planning:**
  - model check the formula on the TS starting from the given initial state.
  - use the path (paths) used in the above model checking for returning the plan.
- *This basic technique works only when we have complete information (or at least total observability on state):*
  - *Sequential plans if initial state known and actions are deterministic*
  - *Conditional plans if many possible initial states and/or actions are nondeterministic*

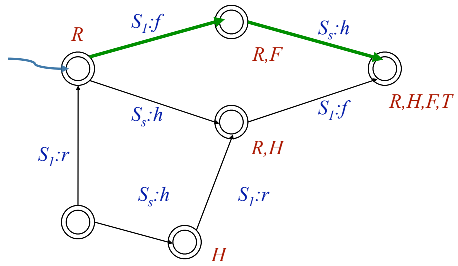


# Example

- Operators (Services + Mappings)
  - Registered ∧ ¬FlightBooked → [$S_1$:bookFlight] FlightBooked
  - ¬Registered → [$S_1$:register] Registered
  - ¬HotelBooked → [$S_2$:bookHotel] HotelBooked
- Additional constraints (Community Ontology):
  - TravelSettledUp ≡ FlightBooked ∧ HotelBooked ∧ EventBooked
- Goals (Client Service Requests):
  - Starting from ***the*** state Registered ∧ ¬FlightBooked ∧ ¬ HotelBooked ∧ ¬EventBooked check <any*>TravelSettedUp
  - Starting from ***all*** states such that ¬FlightBooked ∧ ¬ HotelBooked ∧ ¬EventBooked check <any*>TravelSettledUp



# Example




Plan:
$S_1$:bookFlight;
$S_2$:bookHotel

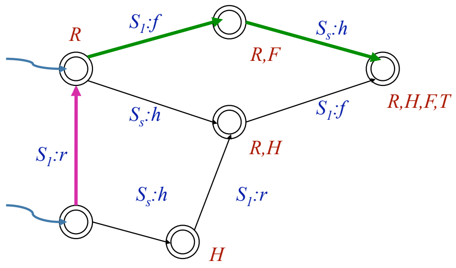
*Starting from the state*

Registered ∧ ¬ FlightBooked ∧ ¬ HotelBooked ∧ ¬ EventBooked

*check*

<any*>TravelSettledUp

## *Example*




Plan:
if(¬Registered) {
$S_1$:register;
}
$S_1$:bookFlight;
$S_2$:bookHotel

*Starting from all states where*

¬ FlightBooked ∧ ¬ HotelBooked ∧ ¬ EventBooked

*check*

<any*>TravelSettledUp



## *Satisfiability*

- Observe that a formula $\Phi$ may be used to select among all TS T those such that for a given state s we have that T,s $\models \Phi$
- SATISFIABILITY: Given a formula $\Phi$ verify whether there exists a TS T and a state s such that. Formally:

  check whether exists T, s such that T,s $\models \Phi$

- Satisfiability is:
  - PSPACE for HennesyMilner Logic
  - EXPTIME for PDL
  - EXPTIME for Mu-Calculus



## *References*

[Stirling Banff96] C. Stirling: Modal and temporal logics for processes. Banff Higher Order Workshop LNCS 1043, 149-237, Springer 1996

[Bradfield&Stirling HPA01] J. Bradfield, C. Stirling: Modal logics and mu-calculi. Handbook of Process Algebra, 293-332, Elsevier, 2001.

[Stirling 2001] C. Stirling: Modal and Temporal Properties of Processes. Texts in Computer Science, Springer 2001

[Kozen&Tiuryn HTCS90] D. Kozen, J. Tiuryn: Logics of programs. Handbook of Theoretical Computer Science, Vol. B, 789–840. North Holland, 1990.

[HKT2000] D. Harel, D. Kozen, J. Tiuryn: Dynamic Logic. MIT Press, 2000.

[Clarke& Schlingloff HAR01] E. M. Clarke, B. Schlingloff: Model Checking. Handbook of Automated Reasoning 2001: 1635-1790

[CGP 2000] E.M. Clarke, O. Grumberg, D. Peled: Model Checking. MIT Press, 2000.

[Emerson HTCS90] E. A. Emerson. Temporal and Modal Logic. Handbook of Theoretical Computer Science, Vol B: 995-1072. North Holland, 1990.

[Emerson Banff96] E. A. Emerson. Automated Temporal Reasoning about Reactive Systems. Banff Higher Order Workshop, LNCS 1043, 111-120, Springer 1996

[Vardi CST] M. Vardi: Alternating automata and program verification. Computer Science Today -Recent Trends and Developments, LNCS Vol. 1000, Springer, 1995.

[Vardi etal CAV94] M. Vardi, O. Kupferman and P. Wolper: An Automata-Theoretic Approach to Branching-Time Model Checking (full version of CAV'94 paper).

[Schneider 2004] K. Schenider: Verification of Reactive Systems, Springer 2004.