Formal Models of Service Behaviors

Logics of Programs

Giuseppe De Giacomo

Service Integration

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

HennessyMilner Logic

HM Logic aka (multi) modal logic Ki

- Syntax:

Φ := Final | P (atomic propositions)
[a]Φ | <a>Φ (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>Φ means there exists an a-transition that leads to a state where Φ holds; i.e., expresses the capability of executing action a bringing about Φ
- [a]Φ means that all a-transitions lead to states where Φ holds; i.e., express that executing action a brings about Φ

HennessyMilner Logic

- Semantics: assigns meaning to the formulas.
- Given a TS T = < A, S, S^0, δ, F>, a state s $\in$ S, and a formula Φ, 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

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

- Another way to give the same semantics to formulas: formulas extension in a transition system assigns meaning to the formulas.

- Given a TS T = < A, S, S^0, δ, F> "the extension of a formula Φ in T", denote by $(\Phi)^T$, is defined as follows:
  - $(Final)^T$ = F (similarly $P^T = \{s \mid s \in P\}$);
  - $([a]\,\Phi)^T$ = $\{s \mid \forall s'.\ s \to_a s'$ implies $s' \in (\Phi)^T\}$;
  - $(\langle a\rangle\Phi)^T$ = $\{s \mid \exists s'.\ s \to_a s'$ 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

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

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

$$T,s \models \Phi \quad \text{or} \quad s \in (\Phi)^T$$

- 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 Φ — *– the query*

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

Examples

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

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

Satisfiability

- Observe that a formula Φ 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 Φ 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

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

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

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

  VAL: check the non existence 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 $\rightarrow$
  ((<10p><small><collect$_s$>Final) $\wedge$ (<20p><big><collect$_b$>Final))
- <10p><small><collect$_s$>Final $\wedge$ [10p]false

Logic of Programs and Bisimulation

- Consider two TS, T = (A,S,s_0,δ, F) and T' = (A,S',t_0,δ', F').
- Let L be the language formed by all HennessyMilner Logic formulas.

- We define:
  - $\sim_L$ = {(s,t) | for all Φ of L we have T,s $\models \Phi$ iff T',t $\models \Phi$}
  - $\sim$ = {(s,t) | 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.

An same results holds also for the PDL and Modal Mu-Calculus introduced below.

Logic of Programs and Bisimulation

Show: s ~ t implies s $\sim_L$ t by structural induction on formulas of L.

Proofs by induction

Show that **property** (s~t) ***is closed wrt the rules of the inductively defined set*** (formation rules for formulas in L)

That is:

- Show **Base Cases** (atomic formulas)
- Show **Recursive Cases** by assuming property holds for smaller cases (**inductive hypothesis**)

Logic of Programs and Bisimulation

Show: s $\sim_L$ t implies s ~ t by coinduction showing that s $\sim_L$ t is a bisimulation.

Proofs by coinduction

Show that **property** (s$\sim_L$t) ***is closed wrt the rules of the coinductively defined set*** (bisimulation s ~ t)

That is:

- Assume **property** holds, show that applying the **recursive rules** it continues to hold.

Notice: no base cases, only recursive cases!!!

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)*

  $r := a \mid r_1 + r_2 \mid r_1;r_2 \mid r^* \mid \Phi?$ *(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 Φ_1 all possible executions of r that terminate reach a state of the TS where Φ_2 holds*

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

- Useful abbereviations:
  - any stands for $(a_1 + \cdots + a_\nu)$ *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 formulas 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 Φ*
- $\mu X.\ \Phi \vee \langle any\rangle X$ *along some run eventually Φ*
- $\nu X.\ [a](\mu Y.\ \langle any\rangle true \wedge [any\text{-}b]Y) \wedge X$
  every run that contains a contains later b

Modal Mu-Calculus

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

- To understand fixpoint operators one has to consider them as fixpoint of equations:
- Namely given $\mu X.\Phi(X)$ and $\nu X.\Phi(X)$ consider the equation

$$X \equiv \Phi(X)$$

Then:
- $\mu X.\Phi(X)$ stands for the smallest predicate X such that $X \equiv \Phi(X)$ – or $\Phi(X) \rightarrow X$
- $\nu X.\Phi(X)$ stands for the largest predicate X such that $X \equiv \Phi(X)$ – or $X \rightarrow \Phi(X)$

Notice:
- $\mu X.\Phi(X)$ is defined by induction and computed by least fixpoint algorithm over the TS
- $\nu X.\Phi(X)$ is defined by coinduction and computed by greatest fixpoint algorithm over the TS

- Examples:
  - ν X. $\Phi \wedge$ [any]X - gfp of X $\equiv \Phi \wedge$ [any]X
  - μ X. $\Phi \vee$ [any]X - lfp of X $\equiv \Phi \vee$ [any]X
  - μ X. $\Phi \vee$ <any>X - lfp of X $\equiv \Phi \vee$ <any>X
  - ν X. [a](μ Y. <any>true $\wedge$ [any-b]Y) $\wedge$ X
    - lfp of y $\equiv$ <any>true $\wedge$ [any-b]Y
    - gfp of X $\equiv$ [a](*lfp above*) $\wedge$ X

Examples of Modal Mu-Calculus

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

- 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*
  - $S_i \models \mu$ X. Final $\vee$ [any] X *eventually a final state is reached*
  - $S_0 \models \nu$ Z. (μ X. Final $\vee$ [any] X) $\wedge$ [any] Z *or equivalently*
    $S_0 \models$ [any*](μ X. Final $\vee$ [any] X) *from everywhere eventually final*

Modal Mu-Calculus extends PDL

We can easily translate in Mu-Calculus all PDL formulas.
Here is the translation function T: PDL → Mu Calculus:

$T(<a>\Phi) = <a>T(\Phi)$
$T(<r_1 + r_2>\Phi) = T(<r_1>\Phi) \vee T(<r_2>\Phi)$
$T(<r_1;r_2>\Phi) = T(<r_1> <r_2>\Phi)$
$T(<r^*> \Phi) = \mu X.\ T(\Phi) \vee T(<r>X)$
$T(<\Phi_1?>\Phi_2) = T(\Phi_1) \wedge T(\Phi_2)$

$T([a]\Phi) = [a]T(\Phi)$
$T([r_1 + r_2]\Phi) = T([r_1]\Phi) \wedge T([r_2]\Phi)$
$T([r_1;r_2]\ \Phi) = T([r_1][r_2]\Phi)$
$T([r^*]\ \Phi) = \nu X.\ T(\Phi) \wedge T([r]X)$
$T([\Phi_1?]\Phi_2) = T(\Phi_1) \rightarrow T(\Phi_2)$

$T(P) = P$
$T(\neg\Phi) = \neg T(\Phi)$
$T(\Phi_1 \wedge \Phi_2) = T(\Phi_1\ ) \wedge T(\Phi_2)$
$T(\Phi_1 \wedge \Phi_2) = T(\Phi_1\ ) \vee T(\Phi_2)$

$T(X) = X$

(although X is not a PDL formula we need this auxiliary definition in the translation)

Notice: no alternation of least and greatest fixpoints!!!

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

Modal Mu-Calculus extends CTL

We can easily translate in Mu-Calculus all CTL formulas.
Here is the translation function T: CTL → Mu Calculus:

CTL formulas:

$\Phi\ := P\ |$ *(atomic propositions)*
$\quad \neg\ \Phi\ |\ \Phi_1 \wedge \Phi_2\ |\ \Phi_1 \vee \Phi_2\ |$ *(boolean operators)*
$\quad EX\Phi\ |\ EF\Phi\ |\ EG\Phi\ |\ \Phi_1\ EU\ \Phi_2|$ *(temporal (modal) operators on a path)*
$\quad AX\Phi\ |\ AF\Phi\ |\ AG\Phi\ |\ \Phi_1\ AU\ \Phi_2$ *(temporal (modal) operators on all paths)*

$T(EX\ \Phi) = <->T(\Phi)$
$T(EF\Phi) = = \mu\ Z.\ T(\Phi) \vee <->Z$
$T(EG\Phi) = = \nu\ Z.\ T(\Phi) \wedge <->Z$
$T(\Phi_1 EU \Phi_2) = = \mu\ Z.\ T(\Phi_2) \vee \Phi_1 \wedge <->Z$

$T(AX\ \Phi) = [-]T(\Phi)$
$T(AF\Phi) = = \mu\ Z.\ T(\Phi) \vee [-]Z$
$T(AG\Phi) = = \nu\ Z.\ T(\Phi) \wedge [-]Z$
$T(\Phi_1 AU \Phi_2) = = \mu\ Z.\ T(\Phi_2) \vee \Phi_1 \wedge [-]Z$

$T(P) = P$
$T(\neg\Phi) = \neg T(\Phi)$
$T(\Phi_1 \wedge \Phi_2) = T(\Phi_1\ ) \wedge T(\Phi_2)$
$T(\Phi_1 \wedge \Phi_2) = T(\Phi_1\ ) \vee T(\Phi_2)$

Notice: no alternation of least and greatest fixpoints!!!

Modal Mu-Calculus extends CTL*

We can translate in Mu-Calculus all CTL* formulas.
Here is the translation function T: CTL → Mu Calculus:

CTL formulas:

Φ := P | *(atomic propositions)*
$\neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid$ *(boolean operators)*
$E\Theta \mid A\Theta$ *(Exist a path/Forall paths)*
where Θ any LTL formula *(LTL temporal formula on a path)*

The translation function is not trivial (the translation may generate an exponential formula).

Important: the resulting formula as at most one alternation of least and greatest fixpoint.

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 nested 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

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.