Programs and Their Semantics

Dynamic Logic, Lecture 1

Igor Sedlár

Institute of Computer Science of the Czech Academy of Sciences

Faculty of Arts, Charles University
Fall Semester 2023-24

Course overview – 1

The course introduces some logics for reasoning about the properties of computer programs (mostly equivalence and correctness). In particular,

- Kleene algebra and
- modal logic (Propositional Dynamic Logic, Linear Temporal Logic).

Some related topics (e.g. finite automata) will be discussed along the way.

Course overview – 2

The fail/pass decision will be based on

- lecture attendance
- solution of 2 problem sets (roughly: mid-Nov and early Jan)

Course materials etc.

- email me: `sedlar@cs.cas.cz`
- course webpage:

Me: Igor Sedlár, Institute of Computer Science CAS. (Pod Vodárenskou věží 271/2, Prague 8. Metro C, Ládví.) `www.cs.cas.cz/sedlar/`

Equivalence of programs: A motivating example

(i)

def $\texttt{print\_primes}(y)$:

$x := 2$
while $x \leq y$ **do**
  if $\texttt{is\_prime}(x)$ **then**
    $\texttt{print}(x)$
    $x := x + 1$
  else
    $x := x + 1$
  end if
end while

(ii)

def $\texttt{print\_primes}(y)$:

$x := 2$
while $x \leq y$ **do**
  if $\texttt{is\_prime}(x)$ **then**
    $\texttt{print}(x)$
  end if
  $x := x + 1$
end while

Figure: Two programs for printing out primes.

A formal language of programs

Propositional variables: $\Pi = \{\mathtt{p}_1, \mathtt{p}_2, \ldots\}$, program (action) variables $\Sigma = \{\mathtt{a}_1, \mathtt{a}_2, \ldots\}$.

Definition 1

Boolean formulas (Fm)

$B, C ::= \mathtt{p} \in \Pi$	
$\mid \top$	*true*
$\mid \bot$	*false*
$\mid \neg B$	*not*
$\mid B \land C$	*and*
$\mid B \lor C$	*or*

Program expressions (Pr)

$E, F ::= \mathtt{a} \in \Sigma$	
$\mid$ **skip**	*"Do nothing" / "Wait"*
$\mid$ **abort**	*"Stop the computation"*
$\mid E; F$	*"Do E, then do F"*
$\mid$ **if** B **then** E **else** F	*Conditionals*
$\mid$ **while** B **do** E	*While loops*

Example ...(i)

$\mathtt{a}; (\textbf{while}\ \mathtt{range}\ \textbf{do}\ (\textbf{if}\ \mathtt{prime}\ \textbf{then}\ \mathtt{print}; \mathtt{inc}\ \textbf{else}\ \mathtt{inc}))$

Relational semantics – 1

Definition 2

A relational model for programs is $M = \langle X, \mathsf{sat}_M, \mathsf{rel}_M \rangle$ *where*

- $X \neq \emptyset$
- $\mathsf{sat}_M : \Pi \to \mathcal{P}(X)$
- $\mathsf{rel}_M : \Sigma \to \mathcal{P}(X \times X)$

sat_M *generalizes to* $Fm \to \mathcal{P}(X)$ *in the usual way.*

Intuition: X is a set of "states"; $\mathsf{sat}_M(\mathtt{p})$ is the set of states where $\mathtt{p}$ "is satisfied" and $\mathsf{rel}_M(\mathtt{a})$ is the "input-output relation" for a ($\langle x, y \rangle \in \mathsf{rel}_M(\mathtt{a})$ iff a may halt in state y when executed in x).

Note: $\mathsf{rel}_M(\mathtt{a})$ is not necessarily a (total) function (non-determinism).

Relational semantics – 2

Example ...(i) again

- $X = \mathbb{N} \times \mathbb{N} \times \overbrace{\mathbb{N}^*}^{\text{finite sequences over } \mathbb{N}}$
- $\mathsf{sat}(\mathtt{range}) = \{\langle n, m, s\rangle \mid n \leq m\}$
- $\mathsf{sat}(\mathtt{prime}) = \{\langle n, m, s\rangle \mid n \text{ is prime}\}$
- and rel is defined so that

$$\langle n, m, s\rangle \xrightarrow{\mathtt{a}} \langle 2, m, s\rangle \qquad \langle n, m, s\rangle \xrightarrow{\mathtt{print}} \langle n, m, sn\rangle$$

$$\langle n, m, s\rangle \xrightarrow{\mathtt{inc}} \langle n+1, m, s\rangle$$

Relational semantics – 3

Recall: If $R, R_1, R_2 \subseteq X \times X$, then

- $R_1 \circ R_2 = \{\langle x, y\rangle \mid \exists z \in X : \langle x, z\rangle \in R_1 \;\;\&\;\; \langle z, y\rangle \in R_2\}$
- $R^* = \bigcup_{n \geq 0} R^n$, where $R^0 = 1_X = \{\langle x, x\rangle \mid x \in X\}$ and $R^{n+1} = R^n \circ R$.

Definition 3

Given M, we define $[\![-]\!]_M : Fm \cup Pr \to \mathcal{P}(X \times X)$:

- $[\![B]\!]_M = 1_{\mathsf{sat}_M(B)}$ *and* $[\![\mathsf{a}]\!]_M = \mathsf{rel}_M(\mathsf{a})$ *for* $B \in Fm$, $\mathsf{a} \in \Sigma$
- $[\![\mathbf{skip}]\!]_M = 1_X$ *and* $[\![\mathbf{abort}]\!]_M = \emptyset$
- $[\![E; F]\!]_M = [\![E]\!]_M \circ [\![F]\!]_M$
- $[\![\mathbf{if}\ B\ \mathbf{then}\ E\ \mathbf{else}\ F]\!]_M = ([\![B]\!]_M \circ [\![E]\!]_M) \cup ([\![\neg B]\!]_M \circ [\![F]\!]_M)$
- $[\![\mathbf{while}\ B\ \mathbf{do}\ E]\!]_M = ([\![B]\!]_M \circ [\![E]\!]_M)^* \circ [\![\neg B]\!]_M$

Programs P and Q are relationally equivalent iff $[\![P]\!]_M = [\![Q]\!]_M$ for all M. (Notation: $P \equiv Q$).

Relational semantics – 4

Simplifying notation:

$$\textbf{skip} = 1 \qquad\qquad \textbf{abort} = 0$$

$$\textbf{if } B \textbf{ then } E \textbf{ else } F = E +_B F \qquad\qquad \textbf{while } B \textbf{ do } E = E^{(B)}$$

We define $\textbf{assert } B := 1 +_B 0$. We usually write "$B$" instead of "$\textbf{assert } B$".

We define $\mathcal{H} : Pr \to Fm$ (the halt predicate):

$\mathcal{H}(\mathsf{a}) := \bot \quad \mathcal{H}(\textbf{skip}) := \top \quad \mathcal{H}(\textbf{abort}) := \bot \quad \mathcal{H}(E;F) := \mathcal{H}(E) \wedge \mathcal{H}(F)$

$\mathcal{H}(E +_B F) := (B \wedge \mathcal{H}(E)) \vee (\neg B \wedge \mathcal{H}(F)) \qquad \mathcal{H}(E^{(B)}) := \neg B$.

Relational semantics – 5

Proposition 1

GKAT axioms are relationally valid:

$$
\begin{array}{lcl}
\text{U1.} & E +_B E \equiv E \\
\text{U2.} & E +_B F \equiv F +_{(\neg B)} E \\
\text{U3.} & (E +_B F) +_C G \equiv E +_{(B \wedge C)} (F +_C G) \\
\text{U4.} & E +_B F \equiv B; E +_B F \\
\text{U5.} & (E +_B F); G \equiv (E; G) +_B (F; G)
\end{array}
\qquad
\begin{array}{lc}
\text{S1.} & E; (F; G) \equiv (E; F); G \\
\text{S2.} & 0; E \equiv 0 \\
\text{S3.} & E; 0 \equiv 0 \\
\text{S4.} & 1; E \equiv E \\
\text{S5.} & E; 1 \equiv E
\end{array}
$$

$$
\begin{array}{lc}
\text{W1.} & E^{(B)} \equiv E; E^{(B)} +_B 1 \\
\text{W2.} & (E +_C 1)^{(B)} \equiv (C; E)^{(B)}
\end{array}
\qquad
\text{W3.} \quad \frac{G \equiv E; G +_B F}{G \equiv E^{(B)}; F} \text{ if } \mathcal{H}(E) = \bot
$$

Example

Hence, $(E; F +_C F)^{(B)} \equiv ((E +_C 1); F)^{(B)}$.

Trace semantics – 1

A state is a complete and consistent set of literals (containing exactly one of $\mathtt{p}$ and $\bar{\mathtt{p}}$ for each $\mathtt{p} \in \Pi$). We write $S \vDash \mathtt{r}$ if $\mathtt{r} \in S$. ($S \vDash B$ as expected.)

Definition 4

A trace (over Π and Σ) is a sequence of the form

$$S_1 \mathtt{a}_1 S_2 \dots \mathtt{a}_{n-1} S_n$$

where $n \geq 1$, each S_i is a state (over Π) and each $\mathtt{a}_j \in \Sigma$. Let Tr be the set of all traces.

Note: If Π is finite, then each state is a word over the set of literals and each trace is a word in $(\mathsf{States} \cdot \Sigma)^* \cdot \mathsf{States}$.

Example

$(\mathtt{on}\,\mathtt{ok})\,\mathtt{switch}\,(\overline{\mathtt{on}}\,\mathtt{ok})\,\mathtt{switch}\,(\mathtt{on}\,\mathtt{ok})\,\mathtt{break}\,(\mathtt{on}\,\overline{\mathtt{ok}})$

Trace semantics – 2

Definition 5

A trace model for programs is $\mathtt{tra} : \Sigma \to Tr$ *where*

$$\mathtt{tra}(\mathtt{a}) \subseteq \{S\mathtt{a}T \mid S, T \textit{ states}\}$$

The canonical trace model is the maximal trace model (i.e. $\mathtt{can}(\mathtt{a}) = \{S\mathtt{a}T \mid S, T \textit{ states}\}$*).*

Fusion product: partial function $\diamond : Tr \times Tr \to Tr$

$$xS \diamond Ty = \begin{cases} xSy & S = T \\ \text{undefined} & S \neq T \end{cases}$$

Lifted to sets of traces: $K \diamond L = \{w \diamond u \mid w \in K \ \ \& \ \ u \in L\}$. We define $K^0 :=$ States, $K^{n+1} = K^n \diamond K$, and $K^* = \bigcup_{n \geq 0} K^n$.

Trace semantics – 3

Definition 6

Given tra, *we define* $[\![-]\!]_{\mathsf{tra}} : Fm \cup Pr \to 2^{Tr}$ *as follows:*

- $[\![B]\!]_{\mathsf{tra}} = \{S \mid S \vDash B\}$ *and* $[\![\mathsf{a}]\!]_{\mathsf{tra}} = \mathsf{tra}(\mathsf{a})$
- $[\![\mathbf{skip}]\!]_{\mathsf{tra}} = \mathsf{States} \quad [\![\mathbf{abort}]\!]_{\mathsf{tra}} = \emptyset$
- $[\![E; F]\!]_{\mathsf{tra}} = [\![E]\!]_{\mathsf{tra}} \diamond [\![F]\!]_{\mathsf{tra}}$
- $[\![\mathbf{if}\ B\ \mathbf{then}\ E\ \mathbf{else}\ F]\!]_{\mathsf{tra}} = ([\![B]\!]_{\mathsf{tra}} \diamond [\![E]\!]_{\mathsf{tra}}) \cup ([\![\neg B]\!]_{\mathsf{tra}} \diamond [\![F]\!]_{\mathsf{tra}})$
- $[\![\mathbf{while}\ B\ \mathbf{do}\ E]\!]_{\mathsf{tra}} = ([\![B]\!]_{\mathsf{tra}} \diamond [\![E]\!]_{\mathsf{tra}})^* \diamond [\![\neg B]\!]_{\mathsf{tra}}$

We denote $[\![-]\!]_{\mathsf{can}}$ *simply as* $[\![-]\!]$.

Trace semantics – 4

Theorem 1

$E \equiv F$ iff $[\![E]\!] = [\![F]\!]$.

Proof (sketch). 1. For each $M = \langle X, \mathsf{sat}_M, \mathsf{rel}_M \rangle$, let $\hat{M} : Tr \to 2^{X \times X}$ such that

$$\hat{M}(S) = \bigcap_{\mathrm{p} \in S} [\![\mathsf{p}]\!]_M \qquad \hat{M}(w\mathsf{a}u) = \hat{M}(w) \circ \mathsf{rel}_M(\mathsf{a}) \circ \hat{M}(u)\,.$$

We denote $\hat{M}(K) = \bigcup_{w \in K} \hat{M}(w)$ and prove $[\![E]\!]_M = \hat{M}([\![E]\!])$ by induction on E.

2. Let $\mathsf{cay} : 2^{Tr} \to 2^{Tr \times Tr}$ where

$$\mathsf{cay}(L) = \{\langle w, w \diamond u \rangle \mid w \in Tr \;\;\&\;\; u \in L\}$$

The function cay is injective. We prove by induction on E that $[\![E]\!]_M = \mathsf{cay}\,([\![E]\!])$ for $M = \langle Tr, \mathsf{sat}_M, \mathsf{rel}_M \rangle$ where $\mathsf{sat}_M(\mathrm{p}) = \{w \mid \langle w, w \rangle \in \mathsf{cay}\,([\![\mathsf{p}]\!])\}$ and $\mathsf{rel}_M(\mathsf{a}) = \mathsf{cay}\,([\![\mathsf{a}]\!])$. □

Completeness – 1

Recall the GKAT axioms:

$$
\begin{array}{ll}
\text{BA.} & E = F \text{ valid in BA} \\
\text{U1.} & E +_B E = E \\
\text{U2.} & E +_B F = F +_{(\neg B)} E \\
\text{U3.} & (E +_B F) +_C G = E +_{(B \wedge C)} (F +_C G) \\
\text{U4.} & E +_B F = B; E +_B F \\
\text{U5.} & (E +_B F); G = (E; G) +_B (F; G)
\end{array}
$$

$$
\begin{array}{ll}
\text{S1.} & E; (F; G) = (E; F); G \\
\text{S2.} & 0; E = 0 \\
\text{S3.} & E; 0 = 0 \\
\text{S4.} & 1; E = E \\
\text{S5.} & E; 1 = E
\end{array}
$$

$$
\begin{array}{ll}
\text{W1.} & E^{(B)} = E; E^{(B)} +_B 1 \\
\text{W2.} & (E +_C 1)^{(B)} = (C; E)^{(B)}
\end{array}
$$

$$
\text{W3.} \quad \frac{G = E; G +_B F}{G = E^{(B)}; F} \text{ if } \mathcal{H}(E) \equiv \bot
$$

Definition 7

A program equation $E = F$ is provable iff it is derivable from the GKAT axioms (using equational logic). Notation: $\vdash E = F$.

Completeness – 2

Let GKAT+UA be the set of GKAT axioms extended with the Uniqueness Axiom of Smolka et al. 2020. We express by $\vdash_{\mathsf{UA}} E = F$ that $E = F$ is provable in GKAT+UA.

Theorem 2

$\vdash_{\mathsf{UA}} E = F$ *iff* $[\![E]\!] = [\![F]\!]$.

Proof is beyond the scope of these lectures; see Smolka et al. 2020. □

Open problem

Give a sound and complete axiomatization of relational equivalence using only standard equational and quasi-equational axioms.

Hoare completeness – 1

Partial correctness: If B holds and E is executed, then C will hold upon termination of E (notation: $\{B\}E\{C\}$).

Hoare logic:

$$\frac{}{\{B\}\mathbf{skip}\{B\}} \qquad \frac{}{\{B\}\mathbf{abort}\{\bot\}} \qquad \frac{\{B\}E\{C\} \;\; \{C\}F\{D\}}{\{B\}E;F\{D\}}$$

$$\frac{\{B \wedge C\}E\{D\} \;\; \{\neg B \wedge C\}F\{D\}}{\{C\}\mathbf{if}\ B\ \mathbf{then}\ E\ \mathbf{else}\ F\{D\}} \qquad \frac{\{B \wedge C\}E\{C\}}{\{C\}\mathbf{while}\ B\ \mathbf{do}\ E\{\neg B \wedge C\}}$$

$$\frac{B' \vDash B \;\; \{B\}E\{C\} \;\; C \vDash C'}{\{B'\}E\{C'\}}$$

Hoare completeness – 2

The following are equivalent:

1. $\{B\}E\{C\}$ is satisfied in M
2. $[\![B; E; C]\!]_M = [\![B; E]\!]_M$
3. $[\![B; E; \bar{C}]\!]_M = [\![\bot]\!]_M$.

Theorem 3

$\vdash B; E; C = B; E$ *iff* $[\![B; E; C]\!] = [\![B; E]\!]$.

Proof (sketch). 1. Soundness: Induction on length of derivation. 2. Completeness: Structural induction on E. □

Notes

- The “logic of programs” introduced in this lecture is a version of Guarded Kleene Algebra With Tests; see (Smolka et al., 2020).
- It is a “propositional variant” of while programs; see (Hoare, 1969) and Ch. 3 of (Apt et al., 2009).
- Our proof of Theorem 1 derives from Kappé’s lecture notes (Kappé, 2022); the argument goes back to (Pratt, 1980).
- (A first-order version of) Hoare logic was introduced by Hoare (1969); the relation of its propositional version to a formalism related to ours is studied by Kozen (2000).
- Theorems 3 and 2 are established in (Smolka et al., 2020).
- The standard completeness problem is discussed in (Kappé et al., 2023).

References

- Krzysztof R. Apt, Frank S. de Boer, and Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. Texts in Computer Science. Springer, 3rd edition, 2009.
- Charles Anthony R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12:576–580, 1969.
- Tobias Kappé. Kleene algebra. Course notes, University of Amsterdam, 2022.
- Dexter Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Logic*, 1(1):60–76, July 2000.
- Tobias Kappé, Todd Schmid, and Alexandra Silva. A Complete Inference System for Skip-free Guarded Kleene Algebra with Tests. In T. Wies, editor, *Programming Languages and Systems. ESOP 2023.*, pages 309–336. Springer Nature Switzerland, 2023.
- Vaughan Pratt. Dynamic algebras and the nature of induction. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pages 22–28, New York, NY, USA, 1980. ACM.
- Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. Guarded Kleene algebra with tests: Verification of uninterpreted programs in nearly linear time. In *Proc. 47th ACM SIGPLAN Symp. Principles of Programming Languages (POPL'20)*, pages 61:1–28, New Orleans, January 2020. ACM.