Semantic embedding in a general logic

- **Program verification involves the proof of mathematical statements**
- **A program verifier will need a theorem prover**
- **One approach: verification condition generation**
- **Another approach: embed program logic in logic of a general prover**
  - **programming logic rules can then be derived**
  - **the full power of the theorem prover will be available for proofs**
  - **the implementation of the rules can be guaranteed to be sound**
  - **easy to soundly extend the logic**
- **Embedding Hoare Logic and other programming logics outlined**
- **First a trivial pedagogical example to illustrate concepts**

Deep Semantic Embedding – a preliminary example

- **Consider the propositional language:**

$$\mathit{wff} \ \texttt{::=}\ \texttt{True} \ \mid\ \texttt{N}\ \mathit{wff} \ \mid\ \texttt{C}\ \mathit{wff}\ \mathit{wff} \ \mid\ \texttt{D}\ \mathit{wff}\ \mathit{wff}$$

- **Can represent *wff*s inside higher order logic**
  - **by values of some type, *wff* say**
  - **we haven't described type definitions**
- **Can define a semantic function, $\mathcal{M}$ say, by recursion:**

$$\begin{aligned}
\mathcal{M}(\texttt{True}) &= T \\
\mathcal{M}(\texttt{N}\ w) &= \neg\mathcal{M}(w) \\
\mathcal{M}(\texttt{C}\ w_1\ w_2) &= \mathcal{M}(w_1) \wedge \mathcal{M}(w_2) \\
\mathcal{M}(\texttt{D}\ w_1\ w_2) &= \mathcal{M}(w_1) \vee \mathcal{M}(w_2)
\end{aligned}$$

  - **$\mathcal{M}$ is a constant of higher order logic of type *wff*$\rightarrow$*bool***
  - **defined via a recursion theorem – details omitted**

Shallow Embedding

- wff `::=` `True` | `N` wff | `C` wff wff | `D` wff wff
- Translate wffs in higher order logic terms
- Suppose $[\![w]\!]$ is the translation of w into higher order logic
- Translation mapping $w \mapsto [\![w]\!]$ not defined inside higher order logic
  - corresponds to informal 'parsing and pretty-printing' conventions
- Let $\leadsto$ mean "is translated to"
- A shallow embedding: is a set of translation rules:

$$
\begin{array}{lcl}
[\![\texttt{True}]\!] & \leadsto & \texttt{"}T\texttt{"} \\
[\![\texttt{N}\ w]\!] & \leadsto & \texttt{"}\neg\texttt{"} \frown [\![w]\!] \\
[\![\texttt{C}\ w_1\ w_2]\!] & \leadsto & [\![w_1]\!] \frown \texttt{"}\wedge\texttt{"} \frown [\![w_2]\!] \\
[\![\texttt{D}\ w_1\ w_2]\!] & \leadsto & [\![w_1]\!] \frown \texttt{"}\vee\texttt{"} \frown [\![w_2]\!]
\end{array}
$$

Deep vs Shallow Embedding

- **Deep and shallow embedding are two ends of a spectrum**
  - some aspects of the semantics formalized inside higher order logic
  - others specified via informal notational conventions
- **Deep embedding: theorems about embedded language provable**
  - for example: $\forall w_1\ w_2 \in \mathit{wff}.\ \mathcal{M}(\mathtt{C}w_1w_2) = \mathcal{M}(\mathtt{NDN}w_1\mathtt{N}w_2)$
  - formalizes more of the embedding, but requires expressive host logic
- **Shallow embedding: only theorems in embedded language provable**
  - quantification over wffs is not expressible
  - less in the logic, and hence the embedding is less demanding
  - so is often easier to support complex notations
- **A shallow embeddings of Hoare specifications will be described**
  - outline how the axioms and rules of Floyd-Hoare logic can be derived
- **Homeier's Sunrise system is a deep embedding of Hoare logic**

Embedding command semantics into higher order logic

- **Commands translate to predicates on pairs of states**

- $C \leadsto c$ **where the meaning of** C **is given by** c

$$c(s_1, s_2) = \begin{cases} \mathtt{T} & \text{if executing } C \text{ in state } s_1 \text{ results in state } s_2 \\ \\ \mathtt{F} & \text{otherwise} \end{cases}$$

- **Example: if** $\mathtt{c}$ **is the predicate denoted by** `WHILE T DO SKIP` **then:**

$$\forall s_1\ s_2.\ \mathtt{c}(s_1, s_2) = \mathtt{F}$$

States

- **Program variables will be represented by strings**
- **The type** $state$ **of states that we use is defined by:**

$$state \ = \ string{\to}num$$

- **State** s **has** $`X`$, $`Y`$, $`Z`$ **bound to** 1, 2, 3; **all other strings bound to** 0
  - $s \ = \ \lambda x.\ if\ x = `X`\ then\ 1\ else\ if\ x = `Y`\ then\ 2\ else\ if\ x = `Z`\ then\ 3\ else\ 0$
- **A more complex type of states is needed for arrays**

Translations represent meaning (denotational semantics)

- E, B, C translate to e, b and c, respectively, where:

$$\begin{array}{lcl} e & : & \mathit{state} \rightarrow \mathit{num} \\ b & : & \mathit{state} \rightarrow \mathit{bool} \\ c & : & \mathit{state} \times \mathit{state} \rightarrow \mathit{bool} \end{array}$$

  represent meanings of expressions and commands

- The embedding translation of $X + 1$ is $\lambda s.\ s`X` + 1$

- The embedding translation of $(X + Y) > 10$ is

$$\lambda s.\ (s`X` + s`Y`) > 10$$

- $[\![E]\!]$ and $[\![B]\!]$ denote the translations (meanings) of E and B

- For example:

$$\begin{array}{lcl} [\![X + 1]\!] & = & \lambda s.\ s`X` + 1 \\ [\![(X + Y) > 10]\!] & = & \lambda s.\ (s`X` + s`Y`) > 10 \end{array}$$

Semantics of Commands

- **A command C translates to a term $[\![C]\!]$ of type $state \times state \to bool$**

- **Define constants as 'constructors' for each kind of command**

- **For example:**

$$[\![\texttt{SKIP}]\!] \;=\; \texttt{Skip}$$

  where the constant `Skip` is defined by:

$$\texttt{Skip}(s_1, s_2) \;=\; (s_1 = s_2)$$

Semantics of other programming constructs

- Define constants `Assign`, `Seq`, `If`, `While`
  - just formalise semantics of language (denotational or operational)
  - details of a denotational semantics in the notes

- $[\![V \texttt{ := } E]\!] \ = \ \texttt{Assign}(\text{`}V\text{`}, [\![E]\!])$
  - $\texttt{Assign} : (\mathit{string} \times (\mathit{state}{\rightarrow}\mathit{num})) \rightarrow (\mathit{state} \times \mathit{state}) \rightarrow \mathit{bool}$

- $[\![C_1 \texttt{;}\ C_2]\!] \ = \ \texttt{Seq}([\![C_1]\!], [\![C_2]\!])$
  - $\texttt{Seq} : ((\mathit{state} \times \mathit{state}{\rightarrow}\mathit{bool}) \times (\mathit{state} \times \mathit{state}{\rightarrow}\mathit{bool})) \rightarrow (\mathit{state} \times \mathit{state}) \rightarrow \mathit{bool}$

- $[\![\texttt{IF } B \texttt{ THEN } C_1 \texttt{ ELSE } C_2]\!] \ = \ \texttt{If}([\![B]\!], [\![C_1]\!], [\![C_2]\!])$
  - $\texttt{If} : ((\mathit{state}{\rightarrow}\mathit{bool}) \times (\mathit{state}{\times}\mathit{state}{\rightarrow}\mathit{bool}) \times (\mathit{state}{\times}\mathit{state}{\rightarrow}\mathit{bool})) \rightarrow (\mathit{state}{\times}\mathit{state}) \rightarrow \mathit{bool}$

- $[\![\texttt{WHILE } B \texttt{ DO } C]\!] \ = \ \texttt{While}([\![B]\!], [\![C]\!])$
  - $\texttt{While} : ((\mathit{state}{\rightarrow}\mathit{bool}) \times (\mathit{state} \times \mathit{state}{\rightarrow}\mathit{bool})) \rightarrow (\mathit{state} \times \mathit{state}) \rightarrow \mathit{bool}$

Example

- **The command**

$$\begin{array}{l} R \ \texttt{:=} \ X\texttt{;} \\ Q \ \texttt{:=} \ 0\texttt{;} \\ \texttt{WHILE} \ Y \leq R \ \texttt{DO} \ (R \ \texttt{:=} \ R - Y\texttt{;} \ Q \ \texttt{:=} \ Q + 1) \end{array}$$

- **translates to (means)**

$$\begin{array}{l} \texttt{Seq} \\ \ (\texttt{Assign}(\text{`}R\text{`}, [\![X]\!]), \\ \ \ \texttt{Seq} \\ \ \ \ (\texttt{Assign}(\text{`}Q\text{`}, [\![0]\!]), \\ \ \ \ \ \texttt{While} \\ \ \ \ \ \ ([\![Y \leq R]\!], \\ \ \ \ \ \ \ \texttt{Seq}(\texttt{Assign}(\text{`}R\text{`}, [\![R - Y]\!], \ \texttt{Assign}(\text{`}Q\text{`}, [\![Q + 1]\!]))))) \end{array}$$

- **Expanding $[\![\cdots]\!]$ notation results in:**

$$\begin{array}{l} \texttt{Seq} \\ \ (\texttt{Assign}(\text{`}R\text{`}, \lambda s.\ s\text{`}X\text{`}), \\ \ \ \texttt{Seq} \\ \ \ \ (\texttt{Assign}(\text{`}Q\text{`}, \lambda s.\ 0), \\ \ \ \ \ \texttt{While} \\ \ \ \ \ \ ((\lambda s.\ s\text{`}Y\text{`} \leq s\text{`}R\text{`}), \\ \ \ \ \ \ \ \texttt{Seq}(\texttt{Assign}(\text{`}R\text{`}, \lambda s.\ s\text{`}R\text{`} - s\text{`}Y\text{`}), \ \texttt{Assign}(\text{`}Q\text{`}, \lambda s.\ s\text{`}Q\text{`} + 1))))) \end{array}$$

Semantics of Partial Correctness

- **A partial correctness specification $\{P\}\ C\ \{Q\}$ translates to:**

$$\forall s_1\ s_2.\ [\![P]\!]\ s_1\ \wedge\ [\![C]\!](s_1, s_2) \Rightarrow [\![Q]\!]\ s_2$$

- **To abbreviate this formula, define a constant `Spec` by:**

$$\texttt{Spec}(p, c, q)\ =\ \forall s_1\ s_2.\ p\ s_1\ \wedge\ c(s_1, s_2) \Rightarrow q\ s_2$$

- **The pre and postconditions P and Q are not just the logical formulae themselves, but are the translations $[\![P]\!]$ and $[\![Q]\!]$**

- **Thus:**

$$\{X = 1\}\ X\ \texttt{:=}\ X + 1\ \{X = 2\}$$

translates to

$$\texttt{Spec}([\![X = 1]\!],\ \texttt{Assign}(`X`, [\![X + 1]\!]),\ [\![X = 2]\!])$$

which is

$$\texttt{Spec}((\lambda s.\ s`X` = 1),\ \texttt{Assign}(`X`, (\lambda s.\ s`X` + 1),\ (\lambda s.\ s`X` = 2))$$

Floyd-Hoare Logic

- **Floyd-Hoare logic can be embedded in higher order logic by regarding the concrete syntax of partial correctness specifications as an abbreviation for the corresponding semantic statements**

- **For example:**

$$\{X = x\}\ X\ \texttt{:=}\ X + 1\ \{X = x + 1\}$$

  can be interpreted as abbreviating:

$$\texttt{Spec}([\![X = x]\!],\ \texttt{Assign}(`X`, [\![X + 1]\!]),\ [\![X = x + 1]\!])$$

- **The translation between the syntactic 'surface structure' and the semantic 'deep structure' is straightforward; it can easily be mechanized with a simple parser and pretty-printer**

- **If partial correctness specifications are interpreted this way then the axioms and rules of Hoare logic can be derived in higher order logic**

Example: SKIP-axiom

- To derive the SKIP-axiom it must be shown for arbitrary P that:

$$\vdash \ \{P\}\ \texttt{SKIP}\ \{P\}$$

which abbreviates:

$$\vdash \ \texttt{Spec}([\![P]\!], \texttt{Skip}, [\![P]\!])$$

- Expanding the definition of Spec, we must show,

$$\vdash \ \forall s_1\ s_2.\ [\![P]\!]\ s_1\ \wedge\ \texttt{Skip}(s_1, s_2) \Rightarrow [\![P]\!]\ s_2$$

- Expanding the definition of Skip, we must show,

$$\vdash \ \forall s_1\ s_2.\ [\![P]\!]\ s_1\ \wedge\ (s_1 = s_2) \Rightarrow [\![P]\!]\ s_2$$

- Using properties of equality:

$$\vdash \ \forall s_1\ s_2.\ [\![P]\!]\ s_1 \Rightarrow [\![P]\!]\ s_1$$

Rules proved as implications

- **Lift $\neg$ and $\wedge$ to predicates:** $\neg p$, $p \wedge q$ **mean** $\lambda s.\ \neg p(s)$, $\lambda s.\ p(s) \wedge q(s)$

- **To derive the sequencing rule prove:**

$$\vdash\ \forall p\ q\ r\ c_1\ c_2.\\ \quad \mathtt{Spec}(p, c_1, q)\ \wedge\ \mathtt{Spec}(q, c_2, r)\ \Rightarrow\ \mathtt{Spec}(p, \mathtt{Seq}(c_1, c_2), r)$$

- **To derive the `IF`-rule prove:**

$$\vdash\ p\ q\ b\ c_1\ c_2.\\ \quad \mathtt{Spec}(p \wedge b, c_1, q)\ \wedge\ \mathtt{Spec}(p \wedge \neg b, c_2, q)\ \Rightarrow\ \mathtt{Spec}(p, \mathtt{If}(b, c_1, c_2), q)$$

- **To derive the `WHILE`-rule prove:**

$$\vdash\ \forall p\ b\ c.\ \mathtt{Spec}(p \wedge b, c, p)\ \Rightarrow\ \mathtt{Spec}(p, \mathtt{While}(b, c), p \wedge \neg b)$$

- **By deriving the Hoare logic rules, we guarantee their soundness**

Other programming logic constructs

- **Variants on Hoare logic:**
  - **VDM-style specifications**
  - **strongest and weakest preconditions**
  - **Dynamic logic**
- **Originally separate programming logics**
- **We will define them in higher order logic**
- **Illustrates benefits of using a general host logic**

VDM-style specifications

- **Used in the Vienna Development Method (VDM)**
  - **introduced a variation on Hoare-style specifications**
- **VDM notation reduces the need for auxiliary logical variables by providing a way of refering to the initial values of variables in post-conditions using 'hooked' variables like** $\overleftarrow{X}$
- **Here's a Hoare-style example:**

$$\{X = x \ \wedge \ Y = y\}\ R{:=}\ X\,;\ X{:=}\ Y\,;\ Y{:=}\ R\ \{Y = x \ \wedge \ X = y\}$$

- **VDM-style version is:**

$$\{\mathtt{T}\}\ R{:=}\ X\,;\ X{:=}\ Y\,;\ Y{:=}\ R\ \{Y = \overleftarrow{X} \ \wedge \ X = \overleftarrow{Y}\}$$

- $\overleftarrow{X}$, $\overleftarrow{Y}$ **denote values of** X, Y **in precondition state**

Aczel's semantics of VDM

- **Postcondition is a binary relation on initial and final states**

- **Regard**

$$\{P[X_1, \ldots, X_n]\}\ C\ \{Q[X_1, \ldots, X_n, \overleftarrow{X_1}, \ldots, \overleftarrow{X_n}]\}$$

  as an abbreviation for

$$\texttt{VDM\_Spec}([\![P[X_1, \ldots, X_n]]\!],\ [\![C]\!],\ [\![Q[X_1, \ldots, X_n, \overleftarrow{X_1}, \ldots, \overleftarrow{X_n}]]\!]_2)$$

  where `VDM_Spec` **is defined by:**

$$\texttt{VDM\_Spec}(p, c, r) = \forall s_1\ s_2.\ p\ s_1\ \wedge\ c(s_1, s_2) \Rightarrow r(s_1, s_2)$$

  and the notation $[\![\ \cdots\ ]\!]_2$ **is defined by:**

$$[\![Q[X_1, \ldots, X_n, \overleftarrow{X_1}, \ldots, \overleftarrow{X_n}]]\!]_2\ =\ \lambda(s_1, s_2).\ Q[s_2\text{‘}X_1\text{‘}, \ldots, s_2\text{‘}X_n\text{‘}, s_1\text{‘}X_1\text{‘}, \ldots, s_1\text{‘}X_n\text{‘}]$$

Aczel's WHILE-rule for total correctness

- **Following rule due to Peter Aczel:**

$$\frac{\vdash\ [P\ \wedge\ B]\ C\ [P\ \wedge\ R] \qquad \vdash\ \texttt{Transitive}\ R \qquad \vdash\ \texttt{Well\_Founded}\ R}{\vdash\ [P]\ \texttt{WHILE}\ B\ \texttt{DO}\ C\ [P\ \wedge\ \neg B\ \wedge\ R^*]}$$

where:

$$\begin{array}{lcl} \texttt{Transitive}\ r & = & \forall s_1\ s_2\ s_3.\ r(s_1, s_2)\ \wedge\ r(s_2, s_3) \Rightarrow r(s_1, s_3) \\ \texttt{Well\_Founded}\ r & = & \neg\exists f : \mathit{num}{\rightarrow}\mathit{state}.\ \forall n.\ r(f(n), f(n+1)) \\ & & \textbf{(no infinite chain: } r(s_0, s_1) \wedge r(s_1, s_2) \wedge \cdots) \end{array}$$

- `Transitive` **and** `Well_Founded` **easily defined in higher order logic**
  - **cannot be defined in first order logic**

Dijkstra's weakest preconditions

- **Idea is to calculate a program to achieve a postcondition**
  - **not a theory of post hoc verification**
- **Non-determinism a key idea in Dijksta's presentation**
  - **start with a non-deterministic high level pseudo-code**
  - **refine to deterministic and efficient code**
- **Weakest preconditions (wp) are for total correctness**
- **Weakest *liberal* preconditions (wlp) for partial correctness**
- **If C is a command and Q a predicate, then informally:**
  - $\texttt{wlp}(C, Q)$ = '*The weakest predicate P such that $\{P\}\ C\ \{Q\}$*'
  - $\texttt{wp}(C, Q)$ = '*The weakest predicate P such that $[P]\ C\ [Q]$*'
- **If P and Q are predicates then $Q \Rightarrow P$ means P is 'weaker' than Q, i.e. everything satisfying Q also satisfies P**

Meaning of weakest preconditions

- **If C is a command and Q a predicate, then informally:**

$$\texttt{wlp}(C, Q) \;=\; \text{`}\textit{The weakest predicate } P \textit{ such that } \{P\}\ Q\ \{Q\}\text{'}$$

- **What is the 'weakest predicate' satisfying a condition**
- **Easy definition in higher order logic:**

$$\vdash\ \texttt{wlp}(c, q) \;=\; \lambda s.\ \forall s'.\ c(s, s') \Rightarrow q\ s'$$

- $\texttt{wlp}(c, q)(s)$ **holds if** s **always leads via** c **to a state satisfying** q
- **The relationship with Hoare's notation is given by:**

$$\vdash\ \texttt{Spec}(p, c, q) \;=\; \forall s.\ p\ s \Rightarrow \texttt{wlp}(c, q)\ s$$

Strongest postconditions

- **Define $\mathtt{sp}(C, P)$ to be 'strongest' Q such that $\{P\}\ C\ \{Q\}$**
  - **partial correctness:** $\{P\}\ C\ \{\mathtt{sp}(C, P)\}$
  - $\{P\}\ C\ \{Q\}$ **equivalent to** $\mathtt{sp}(C, P) \Rightarrow Q$

- **Note that `wlp` goes 'backwards', but `sp` goes 'forwards'**
  - **backwards:** $\{P\}\ C\ \{Q\}$ **if and only if** $P \Rightarrow \mathtt{wlp}(C, Q)$
  - **forwards:** $\{P\}\ C\ \{Q\}$ **if and only if** $\mathtt{sp}(C, P) \Rightarrow Q$

- **Strongest postcondition calculation related to symbolic execution**

Meaning of strongest postconditions

- **If C is a command and P a predicate, then informally:**

$$\texttt{sp}(C,P) \ = \ \text{`}\textit{The strongest predicate } Q \textit{ such that } \{P\}\ Q\ \{Q\}\text{'}$$

- **Easy definition in higher order logic:**

$$\vdash \ \texttt{sp}(c,p) \ = \ \ \lambda s'.\ \exists s.\ p(s) \land c(s,s')$$

- **The relationship with Hoare's notation is given by:**

$$\vdash \ \texttt{Spec}(p,c,q) \ = \ \ \forall s'.\ \texttt{sp}(c,p)(s') \Rightarrow q(s')$$

- **Exercise: prove this from the definitions of** `Spec` **and** `sp`

Dynamic logic

- **Dynamic logic relates Hoare logic to modal logic**
- **Invented by V.R. Pratt based on an idea of R.C. Moore (MIT)**
- **States of computation are thought of as** *possible worlds*
  - **if a command** C **transforms an initial state** s **to a final state** s'
  - **then** s' **is thought of as** *accessible* **from** s **(a concept from modal logic)**
- **Modal logic has** *modal formulae* $\Box q$ **and** $\Diamond q$ **where:**
  - $\Box q$ **is true in** s **if** q **is true in** *all states* **accessible from** s
  - $\Diamond q$ **is true in** s **if if** q **is true in** *some state* **accessible from** s
- **Exercise: convince yourself that** $\Diamond q \;\Leftrightarrow\; \neg\Box\neg q$ **and** $\Box q \;\Leftrightarrow\; \neg\Diamond\neg q$

Modal operators of Dynamic Logic

- **Dynamic logic has operators $[C]$ and $<C>$ for each command C**
  - **$[C]$ is analogous to $\Box$**
  - **$<C>$ is analogous to $\Diamond$**
- **These can be defined on the relation c denoted by C as follows:**

$$\begin{array}{rcl} [c]q & = & \lambda s.\ \forall s'.\ c(s,s') \Rightarrow q\ s' \\ <c>q & = & \neg([c](\neg q)) \end{array}$$

  where $\neg$ is negation lifted pointwise to predicates
- **Relation to weakest preconditions**

$$\vdash\ \mathtt{wlp}(c,q)\ =\ [c]q$$

Conclusion

- **Hoare, VDM, Dijskstra, Dynamic Logic, ... ?**
- **From a semantic embedding perspective its just packaging!**
- **Don't need separate logics, just definitions in a general host logic**
- **Current research in the Lab (not exhaustive!)**
  - **derive Hoare Logic from processor models (Fox, Myreen)**
  - **extend Hoare logic to pointers, concurrency and more (Parkinson et al. ...)**
  - **semantically embed separation logic in higher order logic (Tuerk)**
- **Wednesday:** **guest lecture by Joe Hurd of Galois (`www.galois.com`)**