Background

The Situation Calculus

The situation calculus as presented in (Reiter 2001) is a many-sorted second-order language for representing dynamic worlds. There are three disjoint sorts: $action$ for actions, $situation$ for situations, and $object$ for everything else. A situation calculus language $\mathcal{L}$ has the following components: a constant S_0 denoting the initial situation; a binary function $do(a, s)$ denoting the successor situation to s resulting from performing action a; a binary predicate $s \sqsubseteq s'$ meaning that situation s is a subhistory of situation s'; a binary predicate $Poss(a, s)$ meaning that action a is possible in situation s; a countable set of action functions, e.g., $move(x, y)$; and a countable set of relational fluents, i.e., predicates taking a situation term as their last argument, e.g., $ontable(x, s)$. For simplicity of presentation, we ignore functional fluents in this paper.

We use $\mathcal{L}^-$ to denote the language obtained from $\mathcal{L}$ by removing the sort $situation$ and removing the situation argument from every relational fluent. We call an $\mathcal{L}^-$-formula a pseudo-fluent formula (abbreviated "pff"). Let ϕ be a pff, and s be a situation term. We use $\phi[s]$ to denote the formula obtained from ϕ by restoring s as the situation arguments to all fluents mentioned by ϕ.

Frequently, we are interested only in executable situations, namely, action histories in which it is possible to perform the actions one after the other. This is formalized as follows: $executable(s) \stackrel{def}{=} (\forall a, s^*).do(a, s^*) \sqsubseteq s \supset Poss(a, s^*)$.

Any domain of application is axiomatized by a basic action theory $\mathcal{D}$ with the following components:

1. The foundational axioms for situations.
2. Action precondition axioms, one for each action function A, with syntactic form $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x})[s]$, where $\Pi_A(\vec{x})$ is a pff.
3. Successor state axioms, one for each fluent F, with syntactic form $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a)[s]$, where $\Phi_F(\vec{x}, a)$ is a pff. These embody a solution to the frame problem.
4. Unique names axioms for the primitive actions.
5. An initial database, namely a set of axioms describing S_0.

Golog

The formal semantics of Golog is specified by an abbreviation $Do(\delta, s, s')$, which is inductively defined as follows:

1. Primitive actions: For any action term α,
   $Do(\alpha, s, s') \stackrel{def}{=} Poss(\alpha, s) \wedge s' = do(\alpha, s)$.
2. Test actions: For any pff ϕ,
   $Do(\phi?, s, s') \stackrel{def}{=} \phi[s] \wedge s = s'$.
3. Sequence:
   $Do(\delta_1; \delta_2, s, s') \stackrel{def}{=} (\exists s'').Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$.
4. Nondeterministic choice of two actions:
   $Do(\delta_1 \mid \delta_2, s, s') \stackrel{def}{=} Do(\delta_1, s, s') \vee Do(\delta_2, s, s')$.
5. Nondeterministic choice of action arguments:
   $Do((\pi\, x)\delta(x), s, s') \stackrel{def}{=} (\exists x) Do(\delta(x), s, s')$.
6. Nondeterministic iteration:
   $$Do(\delta^*, s, s') \stackrel{def}{=} (\forall P).\{(\forall s_1)P(s_1, s_1) \wedge (\forall s_1, s_2, s_3)[P(s_1, s_2) \wedge Do(\delta, s_2, s_3) \supset P(s_1, s_3)]\} \supset P(s, s').$$
7. Procedure calls: For any $(n+2)$-ary procedure variable (i.e., predicate variable whose last two arguments are the only ones of sort $situation$) P,
   $Do(P(t_1, \ldots, t_n), s, s') \stackrel{def}{=} P(t_1, \ldots, t_n, s, s')$.
8. Blocks with local procedure declarations: Let Env be an environment, i.e., a set of procedure declarations **proc** $P_1(\vec{v}_1)\ \delta_1$ **endProc**; ... ; **proc** $P_n(\vec{v}_n)\ \delta_n$ **endProc**, where $P_1, \ldots, P_n$ are procedure variables. Then
   $$Do(\{Env; \delta\}, s, s') \stackrel{def}{=} (\forall \vec{P}).[\bigwedge_{i=1}^{n} (\forall \vec{v}_i, s_1, s_2).Do(\delta_i, s_1, s_2) \supset P_i(\vec{v}_i, s_1, s_2)] \supset Do(\delta, s, s').$$
   This says: when $P_1, \ldots, P_n$ are the smallest binary relations on situations that are closed under executing their procedure bodies $\delta_1, \ldots, \delta_n$, then any transition (s, s') obtained by executing the main program δ is a transition for executing $\{Env; \delta\}$.

   Conditionals and loops are defined as abbreviations:

   $\textbf{if}\ \phi\ \textbf{then}\ \delta_1\ \textbf{else}\ \delta_2\ \textbf{fi} \stackrel{def}{=} [\phi?; \delta_1] \mid [\neg\phi?; \delta_2]$,

   $\textbf{while}\ \phi\ \textbf{do}\ \delta\ \textbf{od} \stackrel{def}{=} [\phi?; \delta]^*; \neg\phi?$.

Hoare Logic

The basic formulas of Hoare Logic are constructs of the form $\{p\}\ S\ \{q\}$ (called Hoare triples), where S is a program, and p, q are first-order formulas. The intuitive meaning of $\{p\}\ S\ \{q\}$ is: if p holds before the execution of S and the execution of S terminates, then q holds afterwards. For example, the following are axioms and proof rules of a basic Hoare Logic for programs from a simple Algol-like language.

1. Assignment Axiom
   $$\{p(x/t)\}\ x := t\ \{p\},$$
   where $p(x/t)$ denotes the result of replacing all free occurrences of x in p by t.
2. Composition Rule
   $$\frac{\{p\}\ S_1\ \{r\},\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1; S_2\ \{q\}}.$$
3. **if-then-else** Rule
   $$\frac{\{p \wedge e\}\ S_1\ \{q\},\ \{p \wedge \neg e\}\ S_2\ \{q\}}{\{p\}\ \textbf{if}\ e\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{q\}}.$$
4. **while** Rule
   $$\frac{\{p \wedge e\}\ S\ \{p\}}{\{p\}\ \textbf{while}\ e\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg e\}}.$$
5. Consequence Rule
   $$\frac{p \supset p_1,\ \{p_1\}\ S\ \{q_1\},\ q_1 \supset q}{\{p\}\ S\ \{q\}}.$$

However, Hoare Logic is not complete; see (Apt 1981) for a discussion of the incompleteness results. Cook (1978) circumvented these incompleteness problems by defining the notion of relative completeness. The basic idea was to supply Hoare's system with an oracle which had the ability to answer questions concerning the truths of first-order formulas. In this way, he separated reasoning about programs from reasoning about the underlying domain, in his case, arithmetic.

The Proof System HG

In this section, we present a novel Hoare-style proof system HG for partial correctness of Golog programs.

Syntax and Semantics

A well-formed formula of HG (HG-wff) is either an invariant formula or a Hoare triple, which are defined as follows.

Definition 1 *An invariant formula is a construct of the form $\Box Q$, where Q is a pff; a Hoare triple is a construct of the form $\{Q\}\ \delta\ \{R\}$, where Q and R are pffs, and δ is a Golog program.*

In traditional work on Hoare Logic, the semantics of Hoare triples is defined with respect to an interpretation. In the Golog context, since the semantics of programs is defined by macro-expansion into situation calculus formulas, Hoare triples can be conveniently defined as abbreviations for situation calculus formulas.

Let ϕ be a formula. We use $(\forall).\phi$ to denote the first-order closure of ϕ, i.e., the result of prefixing to ϕ universal quantifiers for all free individual variables in ϕ.

Definition 2 *1.* $\Box Q \stackrel{def}{=} (\forall).executable(s) \supset Q[s]$;
2. $\{Q\}\ \delta\ \{R\} \stackrel{def}{=} (\forall).executable(s) \wedge Q[s] \wedge Do(\delta, s, s') \supset R[s']$.

Intuitively, $\Box Q$ means that Q is true in all executable situations; $\{Q\}\ \delta\ \{R\}$ means that if s is an executable situation satisfying Q and the execution of δ in s leads to a situation s', then s' satisfies R.

Axioms and Proof Rules

Let $\mathcal{D}$ be a basic action theory. The proof system $HG(\mathcal{D})$ is defined as follows. Abbreviations given in parentheses are used to refer to the axioms or rules.

Oracle Axioms

Invariant Oracle Axiom (Inv)

$$\Box Q, \text{ where } \mathcal{D} \models \Box Q.$$

Here we adopt Cook's idea in formulating the notion of relative completeness, and supply our proof system with an oracle which can answer questions concerning whether an invariant formula is entailed by a basic action theory. In this way, we can concentrate on reasoning about programs.

Axioms

1. Effect and Frame Axiom (EF)

$$\{\Phi_F(\vec{x}, A(\vec{y}))\}\ A(\vec{y})\ \{F(\vec{x})\},$$
$$\{\neg\Phi_F(\vec{x}, A(\vec{y}))\}\ A(\vec{y})\ \{\neg F(\vec{x})\},$$

where A is an action function, F is a relational fluent with successor state axiom $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a)[s]$. Note that $\Phi_F(\vec{x}, A(\vec{y}))$ can be simplified using unique names axioms for actions.

2. Fluent-Free Axiom (FF)

$$\{Q\}\ \delta\ \{Q\},$$

where no fluent occurs in Q.

3. Test Action Axiom (TA)

$$\{\phi \supset R\}\ \phi?\ \{R\}.$$

Proof Rules

1. Primitive Action Rule (PAR)

$$\frac{\{Q \wedge \Pi_A(\vec{x})\}\ A(\vec{x})\ \{R\}}{\{Q\}\ A(\vec{x})\ \{R\}},$$

where A is an action function with action precondition axiom $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x})[s]$.

2. Sequence Rule (Seq)

$$\frac{\{Q\}\ \delta_1\ \{S\},\ \{S\}\ \delta_2\ \{R\}}{\{Q\}\ \delta_1;\ \delta_2\ \{R\}}.$$

3. Nondeterministic Action Rule (NA)

$$\frac{\{Q\}\ \delta_1\ \{R\},\ \{Q\}\ \delta_2\ \{R\}}{\{Q\}\ \delta_1 \mid \delta_2\ \{R\}}.$$

4. Nondeterministic Action Argument Rule (NAA)

$$\frac{\{Q\}\ \delta(x)\ \{R\}}{\{Q\}\ (\pi\ x)\ \delta(x)\ \{R\}},$$

where x does not occur free in Q or R.

5. Nondeterministic Iteration Rule (NI)

$$\frac{\{Q\}\ \delta\ \{Q\}}{\{Q\}\ \delta^*\ \{Q\}}.$$

6. Consequence Rule (Cons)

$$\frac{\Box(Q \supset Q_1),\ \{Q_1\}\ \delta\ \{R_1\},\ \Box(R_1 \supset R)}{\{Q\}\ \delta\ \{R\}}.$$

7. Conjunction Rule (Conj)

$$\frac{\{Q_1\}\ \delta\ \{R_1\},\ \{Q_2\}\ \delta\ \{R_2\}}{\{Q_1 \wedge Q_2\}\ \delta\ \{R_1 \wedge R_2\}}.$$

8. Disjunction Rule (Disj)

$$\frac{\{Q_1\}\ \delta\ \{R_1\},\ \{Q_2\}\ \delta\ \{R_2\}}{\{Q_1 \vee Q_2\}\ \delta\ \{R_1 \vee R_2\}}.$$

9. Quantification Rule (Quan)

$$\frac{\{Q(x)\}\ \delta\ \{R(x)\}}{\{\forall x Q(x)\}\ \delta\ \{\forall x R(x)\}},\quad \frac{\{Q(x)\}\ \delta\ \{R(x)\}}{\{\exists x Q(x)\}\ \delta\ \{\exists x R(x)\}},$$

where x does not occur free in δ.

10. Recursion Rule (Rec)

$$\frac{\{\{Q_i\}\ P_i(\vec{v}_i)\ \{R_i\}\}_{i=1}^n \vdash \{\{Q_i\}\ \delta_i\ \{R_i\}\}_{i=1}^n}{\{\{Q_i\}\ \{Env;\ P_i(\vec{v}_i)\}\ \{R_i\}\}_{i=1}^n},$$

where $\{\ \phi_i\ \}_{i=1}^n$ denotes the set $\{\phi_i \mid i = 1, \ldots, n\}$. Intuitively, this rule says that we can infer $\{Q_i\}\,\{Env;\ P_i(\vec{v}_i)\}\,\{R_i\},\ i = 1, \ldots, n$ from the fact that $\{\{Q_i\}\,\delta_i\,\{R_i\}\}_{i=1}^n$ can be proved (using the other proof rules and axioms) from the hypotheses $\{\{Q_i\}\ P_i(\vec{v}_i)\ \{R_i\}\}_{i=1}^n$.

11. Invocation Rule (IK)

$$\frac{\{Q\}\ \delta_i{}^{P_j(\vec{t})}_{\{Env;\,P_j(\vec{t})\}}\ \{R\}}{\{Q\}\ \{Env;\ P_i(\vec{v}_i)\}\ \{R\}},$$

where $\delta_i{}^{P_j(\vec{t})}_{\{Env;\,P_j(\vec{t})\}}$ denotes the result of replacing each procedure call $P_j(\vec{t})$ in δ_i by its contextualized version $\{Env;\ P_j(\vec{t})\}$. Intuitively, to execute $\{Env;\ P_i(\vec{v}_i)\}$ is to execute $\delta_i{}^{P_j(\vec{t})}_{\{Env;\,P_j(\vec{t})\}}$. [1]

12. Substitution Rule (Subs)

$$\frac{\{Q(\vec{x})\}\ \{Env; P_i(\vec{x})\}\ \{R(\vec{x})\}}{\{Q(\vec{x}/\vec{t})\}\ \{Env; P_i(\vec{x}/\vec{t})\}\ \{R(\vec{x}/\vec{t})\}},$$

$$\frac{\{Q(\vec{x})\}\ P(\vec{x})\ \{R(\vec{x})\}}{\{Q(\vec{x}/\vec{t})\}\ P(\vec{x}/\vec{t})\ \{R(\vec{x}/\vec{t})\}},$$

where P is an action function or a procedure variable, and $Q(\vec{x}/\vec{t})$ denotes the result of simultaneously substituting terms from $\vec{t}$ for the corresponding variables from $\vec{x}$ in Q.

The following are derived rules:

1. **If** Rule

$$\frac{\{Q \wedge \phi\}\ \delta_1\ \{R\},\ \{Q \wedge \neg\phi\}\ \delta_2\ \{R\}}{\{Q\}\ \mathbf{if}\ \phi\ \mathbf{then}\ \delta_1\ \mathbf{else}\ \delta_2\ \mathbf{fi}\ \{R\}}.$$

2. **While** Rule

$$\frac{\{Q \wedge \phi\}\ \delta\ \{Q\}}{\{Q\}\ \mathbf{while}\ \phi\ \mathbf{do}\ \delta\ \mathbf{od}\ \{Q \wedge \neg\phi\}}.$$

Provability

Due to the recursion rule, the system $HG(\mathcal{D})$ is not a standard proof system. Let $BH(\mathcal{D})$ denote $HG(\mathcal{D})$ without the recursion rule. We first define provability in $BH(\mathcal{D})$, and then use it to define provability in $HG(\mathcal{D})$. In the sequel, we use Φ and Ψ to denote finite sets of HG-wffs.

Definition 3 *A formal proof of Ψ from Φ in $BH(\mathcal{D})$ is a finite sequence S of HG-wffs, each of which is either an axiom of $BH(\mathcal{D})$, an element of Φ, or is obtained from previous formulas of S by a proof rule of $BH(\mathcal{D})$. We write $\Phi \vdash_{BH(\mathcal{D})} \Psi$, if there is a proof of Ψ from Φ in $BH(\mathcal{D})$.*

[1] Note that this rule supports the compositional proof of properties of procedures. For example, suppose that P_1 only calls itself, and P_2 only calls P_1. We can first use the recursion rule to prove the property of P_1, and then use this property and the invocation rule to prove the property of P_2. Without the invocation rule, we can only prove properties of all procedures simultaneously.

Definition 4 *That Ψ is provable in $HG(\mathcal{D})$, written $\vdash_{HG(\mathcal{D})} \Psi$, is inductively defined as follows:*

1. $\vdash_{HG(\mathcal{D})} \varnothing$;

2. If $\vdash_{HG(\mathcal{D})} \Phi$, and $\Phi \vdash_{BH(\mathcal{D})} \Psi$, then $\vdash_{HG(\mathcal{D})} \Psi$;

3. If $\vdash_{HG(\mathcal{D})} \Phi$, and $\Phi \cup \{\{Q_i\}\ P_i(\vec{v}_i)\ \{R_i\}\}_{i=1}^n \vdash_{BH(\mathcal{D})} \{\{Q_i\}\ \delta_i\ \{R_i\}\}_{i=1}^n$, then $\vdash_{HG(\mathcal{D})} \{\{Q_i\}\ \{Env;\ P_i(\vec{v}_i)\}\ \{R_i\}\}_{i=1}^n$.

Example: A Blocks World

In this section, we demonstrate the use of our proof system by proving properties of robot programs in a simple domain: a blocks world. Despite its simplicity, this domain illustrates some important issues in verification of robot programs.

Action Precondition Axioms

$Poss(move(x, y), s) \equiv clear(x, s) \wedge clear(y, s) \wedge x \neq y,$
$Poss(moveToTable(x), s) \equiv$
$\quad clear(x, s) \wedge \neg ontable(x, s).$

Successor State Axioms

$on(x, y, do(a, s)) \equiv a = move(x, y) \vee$
$\quad on(x, y, s) \wedge a \neq moveToTable(x) \wedge$
$\qquad \neg(\exists z) a = move(x, z),$
$above(x, y, do(a, s)) \equiv$
$\quad (\exists z)\{a = move(x, z) \wedge [z = y \vee above(z, y, s)]\} \vee$
$\quad above(x, y, s) \wedge a \neq moveToTable(x) \wedge$
$\qquad \neg(\exists z) a = move(x, z),$
$clear(x, do(a, s)) \equiv$
$\quad (\exists y)\{[(\exists z) a = move(y, z) \vee a = moveToTable(y)] \wedge$
$\qquad on(y, x, s)\} \vee$
$\quad clear(x, s) \wedge \neg(\exists y) a = move(y, x),$
$ontable(x, do(a, s)) \equiv a = moveToTable(x) \vee$
$\quad ontable(x, s) \wedge \neg(\exists y) a = move(x, y).$

Initial Database

$\phi[S_0]$, where $\phi \in \mathcal{A}_{bw}$, which is the set of the following pffs:
$on(x, y) \equiv above(x, y) \wedge \neg(\exists z)(above(x, z) \wedge above(z, y)),$
$clear(x) \equiv \neg(\exists y) on(y, x),$
$ontable(x) \equiv \neg(\exists y) on(x, y),$
$\neg above(x, x),$
$above(x, y) \wedge above(y, z) \supset above(x, z),$
$above(x, y) \wedge above(x, z) \supset$
$\quad y = z \vee above(y, z) \vee above(z, y),$
$above(y, x) \wedge above(z, x) \supset$
$\quad y = z \vee above(y, z) \vee above(z, y),$
$ontable(x) \vee (\exists y)(above(x, y) \wedge ontable(y)),$
$clear(x) \vee (\exists y)(above(y, x) \wedge clear(y)),$
$above(x, y) \supset (\exists z) on(x, z) \wedge (\exists w) on(w, y).$

Cook and Liu (2002) show that $\mathcal{A}_{bw}$ is complete in the following sense: if we model a state of blocks world by a finite collection of finite chains, then every sentence that is true in all such models is a consequence of $\mathcal{A}_{bw}$.

Let $\mathcal{D}_{bw}$ denote the basic action theory of this blocks world. We can prove that for each $\phi \in \mathcal{A}_{bw}$, $\mathcal{D}_{bw} \models \Box\phi$.

While Loop

Consider the following Golog program β, which nondeterministically moves a block onto another block, so long as there are at least two blocks on the table:

$$\textbf{while } (\exists x, y)[ontable(x) \wedge ontable(y) \wedge x \neq y] \textbf{ do } (\pi\, u, v) move(u, v) \textbf{ od}$$

We want to prove that whenever this program terminates, there is a unique block on the table, provided there was some block on the table to begin with:

$$\{(\exists x) ontable(x)\}\ \beta\ \{(\exists ! y) ontable(y)\}.$$

A proof consists of a sequence of lines. To justify a new line, we annotate it by an axiom or a proof rule, together with the lines that are used as rule premises.

In the following proof, to reduce length of formulas, we use $\phi(x, y)$ to denote $ontable(x) \wedge ontable(y) \wedge x \neq y$.

1. $\{ontable(x) \wedge u \neq x\}$ $move(u, v)$ $\{ontable(x)\}$ — EF
2. $\{\phi(x, y) \wedge u \neq x\}$ $move(u, v)$ $\{ontable(x)\}$ — Cons(1)
3. $\{ontable(y) \wedge u \neq y\}$ $move(u, v)$ $\{ontable(y)\}$ — EF
4. $\{\phi(x, y) \wedge u = x\}$ $move(u, v)$ $\{ontable(y)\}$ — Cons(3)
5. $\{\phi(x, y)\}$ $move(u, v)$ $\{ontable(x) \vee ontable(y)\}$ — Disj(2,4)
6. $\{\phi(x, y)\}$ $(\pi\, u, v) move(u, v)$ $\{ontable(x) \vee ontable(y)\}$ — NAA(5)
7. $\{(\exists x, y)\phi(x, y)\}$ $(\pi\, u, v) move(u, v)$ $\{(\exists x, y)[ontable(x) \vee ontable(y)]\}$ — Quan(6)
8. $\{(\exists x) ontable(x) \wedge (\exists x, y)\phi(x, y)\}$ $(\pi\, u, v) move(u, v)$ $\{(\exists x) ontable(x)\}$ — Cons(7)
9. $\{(\exists x) ontable(x)\}$ β $\{(\exists ! y) ontable(y)\}$ — While(8)

Recursive Procedure

Consider the following Golog procedure which puts all the blocks in the tower with top block b onto the table:

$$\begin{array}{l}\textbf{proc } flattenTower(b) \\ \quad ontable(b)? \mid \\ \quad (\pi\, c)[on(b, c)?;\ moveToTable(b);\ flattenTower(c)] \\ \textbf{endProc}.\end{array}$$

We want to prove its partial correctness:

$$\{x = b \vee above(b, x)\}\ \{Env; flattenTower(b)\}\ \{ontable(x)\}.$$

We will prove a stronger statement:

$$\{ontable(x) \vee x = b \vee above(b, x)\}\ \{Env; flattenTower(b)\}\ \{ontable(x)\}.$$

In what follows, we use $\psi(b, x)$ to denote $ontable(x) \vee x = b \vee above(b, x)$, and $\gamma(b, c)$ to denote $on(b, c)?;\ moveToTable(b);\ flattenTower(c)$.

By the recursion rule, it suffices to prove that

$$\{\psi(b, x)\}\ flattenTower(b)\ \{ontable(x)\} \vdash_{BP(\mathcal{D}_{bw})} \{\psi(b, x)\}\ ontable(b)? \mid (\pi\, c)\gamma(b, c)\ \{ontable(x)\}.$$

We use blank lines to break the proof into paragraphs:

1. $\{\psi(b, x)\}$ $flattenTower(b)$ $\{ontable(x)\}$ — Hypothesis
2. $\{\psi(c, x)\}$ $flattenTower(c)$ $\{ontable(x)\}$ — Subs(1)
3. $\{ontable(x)\}$ $flattenTower(c)$ $\{ontable(x)\}$ — Cons(2)
4. $\{x = c \vee above(c, x)\}$ $flattenTower(c)$ $\{ontable(x)\}$ — Cons(2)

5. $\{\psi(b, x)\}$ $ontable(b)?$ $\{\psi(b, x) \wedge ontable(b)\}$ — TA
6. $\Box\{\psi(b, x) \wedge ontable(b) \supset ontable(x)\}$ — Inv
7. $\{\psi(b, x)\}$ $ontable(b)?$ $\{ontable(x)\}$ — Cons(5,6)

8. $\{ontable(x)\}$ $on(b, c)?$ $\{ontable(x)\}$ — TA
9. $\{ontable(x)\}$ $moveToTable(b)$ $\{ontable(x)\}$ — EF
10. $\{ontable(x)\}$ $\gamma(b, c)$ $\{ontable(x)\}$ — Seq(8,9,3)

11. $\{x = b\}$ $on(b, c)?$ $\{x = b\}$ — FF
12. $\{x = b\}$ $moveToTable(b)$ $\{x = b\}$ — FF
13. $\{true\}$ $moveToTable(b)$ $\{ontable(b)\}$ — EF
14. $\{x = b\}$ $moveToTable(b)$ $\{x = b \wedge ontable(b)\}$ — Conj(12,13)
15. $\{x = b\}$ $moveToTable(b)$ $\{ontable(x)\}$ — Cons(14)
16. $\{x = b\}$ $\gamma(b, c)$ $\{ontable(x)\}$ — Seq(11,15,3)

17. $\{above(b, x)\}$ $on(b, c)?$ $\{above(b, x) \wedge on(b, c)\}$ — TA
18. $\Box\{above(b, x) \wedge on(b, c) \supset x = c \vee above(c, x) \wedge b \neq c\}$ — Inv
19. $\{above(b, x)\}$ $on(b, c)?$ $\{x = c \vee above(c, x) \wedge b \neq c\}$ — Cons(17,18)
20. $\{x = c\}$ $moveToTable(b)$ $\{x = c\}$ — FF
21. $\{above(c, x) \wedge b \neq c\}$ $moveToTable(b)$ $\{above(c, x)\}$ — EF
22. $\{x = c \vee above(c, x) \wedge b \neq c\}$ $moveToTable(b)$ $\{x = c \vee above(c, x)\}$ — Disj(20,21)
23. $\{above(b, x)\}$ $\gamma(b, c)$ $\{ontable(x)\}$ — Seq(19,22,4)

24. $\{\psi(b, x)\}$ $\gamma(b, c)$ $\{ontable(x)\}$ — Disj(10,16,23)
25. $\{\psi(b, x)\}$ $(\pi\, c)\gamma(b, c)$ $\{ontable(x)\}$ — NAA(24)
26. $\{\psi(b, x)\}$ $ontable(b)? \mid (\pi\, c)\gamma(b, c)$ $\{ontable(x)\}$ — NA(7,25)

Soundness and Completeness Results

In traditional work on Hoare Logic, the soundness and completeness results explore the relationship between $I \models \phi$ and $H(I) \vdash \phi$, where I is an interpretation, ϕ is a Hoare triple, and $H(I)$ is a Hoare-style proof system with some set of formulas true in I taken as additional axioms. In our work, the soundness and completeness results will explore the relationship between $\mathcal{D} \models \phi$ and $G(\mathcal{D}) \vdash \phi$, where $\mathcal{D}$ is a basic action theory, ϕ is a Hoare triple, and $G(\mathcal{D})$ is a Hoare-style proof system with some set of formulas entailed by $\mathcal{D}$ taken as additional axioms.

Theorem 5 Total Soundness of HG**.** *For every basic action theory* $\mathcal{D}$*, if* $\vdash_{HG(\mathcal{D})} \Psi$*, then* $\mathcal{D} \models \Psi$*.*

The following are two important lemmas for the theorem. The fixpoint lemma handles the invocation rule, and the induction principle deals with the recursion rule.

Lemma 6 Fixpoint Lemma. *Let* $i = 1, \ldots, n$*. The following is a valid sentence:*

$$(\forall).Do(\{Env;\ P_i(\vec{v}_i)\}, s, s') \equiv Do(\delta_i\,{}^{P_j(\vec{t})}_{\{Env;\ P_j(\vec{t})\}}, s, s').$$

Lemma 7 Induction Principle for Recursive Procedures. *Let* $Q_1, \ldots, Q_n, R_1, \ldots, R_n$ *be pffs. The following is a valid second-order sentence:*

$$(\forall \vec{P})[\bigwedge_{i=1}^{n}\{Q_i\}\ P_i(\vec{v}_i)\ \{R_i\} \supset \bigwedge_{i=1}^{n}\{Q_i\}\ \delta_i\ \{R_i\}] \supset \bigwedge_{i=1}^{n}\{Q_i\}\ \{Env; P_i(\vec{v}_i)\}\ \{R_i\}.$$

The notion of completeness applicable to HG is that of relative completeness. This is because HG has oracle axioms, which are not necessarily recursive. The relative completeness of HG remains open. Here we prove relative completeness of a subsystem of HG. Let WG be the set of Golog programs without procedures. Let HW be the restriction of HG to WG, i.e., programs appearing in Hoare triples are procedureless, and the recursion, invocation and substitution rules concerning procedures are removed. We will prove relative completeness of HW. We first define the notion of expressiveness, which is adapted from that in (Cook 1978).

Definition 8 *Let* $\mathcal{D}$ *be a basic action theory in language* $\mathcal{L}$ *and* Δ *be a set of Golog programs. We say that* $\mathcal{L}$ *is expressive relative to* $\mathcal{D}$ *and* Δ *if for any program* $\delta \in \Delta$ *and any pff* R*, there exists a pff* Q *such that*

$$\mathcal{D} \models (\forall).Q[s] \equiv (\forall s').Do(\delta, s, s') \supset R[s'];$$

we call Q *the weakest liberal precondition of* δ *wrt* R*.*

We first give an example of non-expressiveness by showing that the language of blocks world is not expressive relative to $\mathcal{D}_{bw}$ and WG. It is easy to write a program $\delta \in WG$ which makes a tower of even height with at most one block not in the tower. Then the weakest liberal precondition of δ wrt $(\exists! x) ontable(x)$ would assert that there are an even number of blocks, which we know is not expressible in the language of blocks world.

Theorem 9 Relative Completeness of HW**.** *For any basic action theory* $\mathcal{D}$ *in* $\mathcal{L}$ *such that* $\mathcal{L}$ *is expressive relative to* $\mathcal{D}$ *and* WG*, for any Hoare triple* $\{Q\}\,\delta\,\{R\}$ *such that* $\delta \in WG$*, if* $\mathcal{D} \models \{Q\}\,\delta\,\{R\}$*, then* $\vdash_{HW(\mathcal{D})} \{Q\}\,\delta\,\{R\}$*.*

Here we conjecture a sufficient condition for expressiveness relative to WG. For any basic action theory $\mathcal{D}$ in language $\mathcal{L}$, define $\mathcal{L}^+$ as the extension of $\mathcal{L}$ which contains a sort nat for natural numbers, the language of Peano arithmetic, and two function symbols (their intended interpretations are codings of objects and situations into natural numbers); define $\mathcal{D}^+ = \mathcal{D} \cup \mathcal{P} \cup \mathcal{C}$, where $\mathcal{P}$ is the second-order axiomatization of Peano arithmetic, and $\mathcal{C}$ asserts that the objects and situations are countable. We call $\mathcal{D}^+$ an arithmetical basic action theory. A nice property of models of $\mathcal{D}^+$ is that by using Gödel's β-function we can encode finite sequences of elements from the domain by a pair of natural numbers. We say that $\mathcal{D}^+$ is finite-change if there are only finitely many fluents, and in each model of $\mathcal{D}^+$, each primitive action can only change the value of fluents at finitely many points. We conjecture that if $\mathcal{D}^+$ is finite-change, then $\mathcal{L}^+$ is expressive relative to $\mathcal{D}^+$ and WG.

Conclusions

Golog is a novel logic programming language for high-level robotic control. To establish a systematic approach for proving correctness of robot controllers written in Golog, we have explored Hoare's axiomatic approach to program verification in the Golog context. This is not a routine task due to the differences between Golog and Algol-like languages. Our technical contributions include the definition of the semantics of Hoare triples, and the formulation of the notions of soundness, completeness and expressiveness in the Golog context.

In summary, we have presented a novel Hoare-style proof system for partial correctness of Golog programs. We have proved total soundness of the proof system, and relative completeness of a subsystem of it for procedureless Golog programs. Using this proof system, we can obtain structured and compositional proofs for properties of Golog programs. An important future research topic is to investigate the completeness issue for procedures.

Acknowledgments

I thank Hector Levesque and Ray Reiter for many helpful discussions about this paper. I am grateful to Stephen Cook for his valuable help with this work. Thanks also to the anonymous referees for useful comments.

References

Apt, K. 1981. Ten years of Hoare's logic: a survey–Part I. *ACM Trans. Program. Lang. Syst.* 3(4):431–483.

Apt, K. 1984. Ten years of Hoare's logic: a survey–Part II: nondeterminism. *Theoret. Comput. Sci.* 28:83–109.

Cook, S., and Liu, Y. 2002. A complete axiomatization for blocks world. In *Seventh International Symposium on Artificial Intelligence and Mathematics*.

Cook, S. 1978. Soundness and completeness of an axiom system for program verification. *SIAM J. of Computing* 7(1):70–90.

Hoare, C. 1969. An axiomatic basis for computer programming. *Comm. ACM* 12(10):576–580, 583.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1997. Golog: A logic programming language for dynamic domains. *J. of Logic Programming* 31:59–84.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. Edinburgh University Press. 463–502.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.