FROM ACTION THEORIES TO AGENTS

Chitta Baral

Department of Computer Science
University of Texas at El Paso
El Paso, TX 79902, USA
chitta@cs.utep.edu

http://cs.utep.edu/chitta/chitta.html

August 1997

ACTION THEORIES

Simple action theory

A simple action theory consists of the following:

- Action description
  - Laws
    - Causal relation between actions and fluents, and executability conditions for actions.
      (ex: *driving to the airport* causes *at_airport* to be *true*)
      (ex: *driving to the airport* is executable if *has_car* is *true*)
    - Relationship (often causal) between fluents and fluents
      (May result in indirect effects and/or executability conditions)
      (ex: *at_airport* and *at_home* can not be true at the same time.)
  - Observations
    - Fluent values at different situations (or time points)
      (ex: initially *at_home* and *has_car*)

  * Action occurrences at different situations (or time points)
    (ex: *car_got_hit* occurs at s_1)
  * Temporal ordering between situations (or time points)
    (ex: s_0 preceeds s_1)

- Queries: A *simple query* is of the form:
  Does a formula of fluents holds after the execution of a sequence of actions in the current situation?

  (ex: Is it true that if the agent drives the car in the current situation, he/she/it will be at the airport?)

Example

$$D_0 \left\{ \begin{array}{l} LAWS: \\ \\ \underline{Effects\ of\ actions:} \\ rent\ \mathbf{causes}\ \ has\_car \\ car\_got\_hit\ \mathbf{causes}\ \ \neg has\_car \\ drive\_to\_airport\ \mathbf{causes}\ \ at\_airport \\ pack\ \mathbf{causes}\ \ packed\ \mathbf{if}\ \ at\_home \\ \\ \underline{Executability\ Conditions:} \\ \ \mathbf{executable}\ \ drive\_to\_airport\ \mathbf{if}\ \ has\_car \\ \\ \underline{Relationship\ between\ fluents:} \\ \ \mathbf{always}\ \ \neg(at\_home \wedge at\_airport) \\ \\ OBSERVATIONS: \\ at\_home\ \mathbf{at}\ \ s_0 \\ has\_car\ \mathbf{at}\ \ s_0 \\ \neg packed\ \mathbf{at}\ \ s_0 \end{array} \right.$$

Query1: $(packed \wedge at\_airport)$ **after** $pack, drive\_to\_airport$

Ans: Yes! (i.e. $D_0 \models Query1$)

$$D_1 \begin{cases} OBSERVATIONS: \\ pack \ \textbf{occurs\_at} \ \ s_0 \\ s_0 \ \textbf{precedes} \ \ s_1 \\ car\_got\_hit \ \textbf{occurs\_at} \ \ s_1 \end{cases}$$

Query2: $(packed \wedge at\_airport)$ **after** $drive\_to\_airport$
Ans: No! (i.e. $D_0 \cup D_1 \not\models Query2$)

Query3: $(packed \wedge at\_airport)$ **after** $rent, drive\_to\_airport$
Ans: Yes! (i.e. $D_0 \cup D_1 \models Query3$)

Research in Reasoning about actions

- AIM:
  Designing knowledge representation languages (syntax and semantics) to express action specifications (laws and observations) and queries; and *defining the entailment* (models and procedural methods) between them.
  (Frame problem, ramification, qualification etc. are considered when defining the entailment)
- Specification aspects of recent interest:
  - Actions with non-deterministic (probabilistic) effects,
  - concurrent actions,
  - narratives,
  - actions with duration,
  - natural events,
  - causal constraints,
  - sensing actions.
- Expressive Queries:
  Does a *temporal formula* holds during the execution of a *plan*, which could be a program of actions, involving loops, and conditions.

Action description languages: 2 approaches

- Explicit languages (EL)
  - a high level language with its own semantics
  - oftne with a translation to a logical language
  - Example: $\mathcal{A}$, $\mathcal{AR}$, $\mathcal{L}$; $\mathcal{E}$; Sandewal's languages.
- Implicit languages (IL)
  - Specified in a *particular subclass* of a logical language.
  - Specifications are subjected to *compilations* and/or *minimizations*.
  - Example: Reiter et al; Shanahan and Miller; etc.
- Comparing them
  - The *particular subclass* in an IL corresponds to the syntax of an EL.
  - The compilation and/or minimization in an IL corresponds to the semantics of an EL.
  - EL $\setminus$ independent semantics = IL
    But, independent semantics adds confidence.
  - IL may have more general queries; but not all may be useful or meaningful.

AGENT ARCHITECTURES

Agent in a static world

- Observe the world.
- Plan to achieve the goal.
- Execute the plan.

Deliberative agent in a dynamic world

Repeat until the goal is reached

- Observe and assimilate observation with the current knowledge about the world.

- (Re)plan to achieve the goal from the current situation.

- Execute part of the plan.

Reactive agent (Single level) in a dynamic world

Simple reactive rule:

if $p_1, \ldots, p_n$ **then** a

Example: **if** $obstacle\_in\_front$ **then** $move\_right$

Simple Control module:

A collection of simple reactive rules.
Control Architecture

1. Observe and update fluent values.
2. Find a reactive rule in the control module whose LHS evaluates true w.r.t. the current fluent values.
3. If the RHS of that rule is HALT then stop, else execute the action in the RHS of that rule and go to Step 1.

Traditional hybrid agent in a dynamic world

- Has multiple levels.
- Lower levels are reactive.
- Higher levels are deliberative.

Hybrid agent in a dynamic world: My notion

- Has multiple levels.
  (The action in the RHS of a reactive rule may itself be defined by another control module.)
- No requirement about lower levels being reactive or higher levels being deliberative.
- A reactive control module may not have rules for all possible states. (Avoids the problem of considering exponential number of states.) *I.e. It is not a universal plan.*
- A reactive control module has control rules for the most plausible or the most important or the most critical states.

- For all other states the agent switches back to on-line deliberation to find a plan to *one of the accounted for states (not for the goal state)* and may add compilations of this plan to the reactive control module.

Analogies:

- Cache memory (reactive) and main memory (deliberative).
- Driving: When lost, *finding the path to a known landmark.*

RELATING ACTION THEORIES AND AGENT ARCHITECTURES

Review and Next step

- For agents in a static world, the planning (from the initial situation) part involves action theory.
  A simple action theory without narratives is adequate.
- For deliberative agents in a dynamic world, the planning (from the current situation) part involves action theory.
  The action theory should allow expression of action execution (by the agent), and observations by the agent, and we should be able to plan from the current situation.
- **But, What is the relation between the reactive control architecture and action theories?**

  Hints:
  - Reactive rules have fluents in the LHS and actions in the RHS.
  - The action theory describes the effect of actions (the ones in the RHS) on the fluents, and the relationship between the fluents.
  - The action theory could be from the agents perspective, not from the point of view of a neutral observer.

Intuitive Answers:

- Correctness:
  Perhaps the reactive control module can be shown to be 'correct' w.r.t. to achieving (or maintaining) a goal from a set of initial states w.r.t. a given action theory about the agent and the environment.
- Compilation:
  Perhaps the reactive control module can be automatically constructed (OFF-LINE) so as to achieve (or maintain) a goal from a set of initial states w.r.t. a given action theory about the agent and the environment.

Additional analogies:

- Parsing tables (reactive control) and language theory (action theory).
- Fire drill
- Contingency plan for a flood or a nuclear attack.
- Reacting to common scenarios, and Hmm! let me think to questions that are encountered for the first time.
- Difficulty in recognizing the differences between the pigs that play the role of Babe!

FORMAL CHARACTERIZATION OF SIMPLE REACTIVE MODULES

Control rules and Control modules

- *Simple control rule*:

  if $p_1, \ldots, p_k$ **then** $a_1, \ldots, a_l$

  $p_1, \ldots, p_k$ are fluent literals and $a_1, \ldots, a_l$ are actions.

- *Termination control rule*:

  if $p_1, \ldots, p_k$ **then** $HALT$,

- *Suspension control rule* is of the form

  if $p_1, \ldots, p_k$ **then** $SUSPEND$,

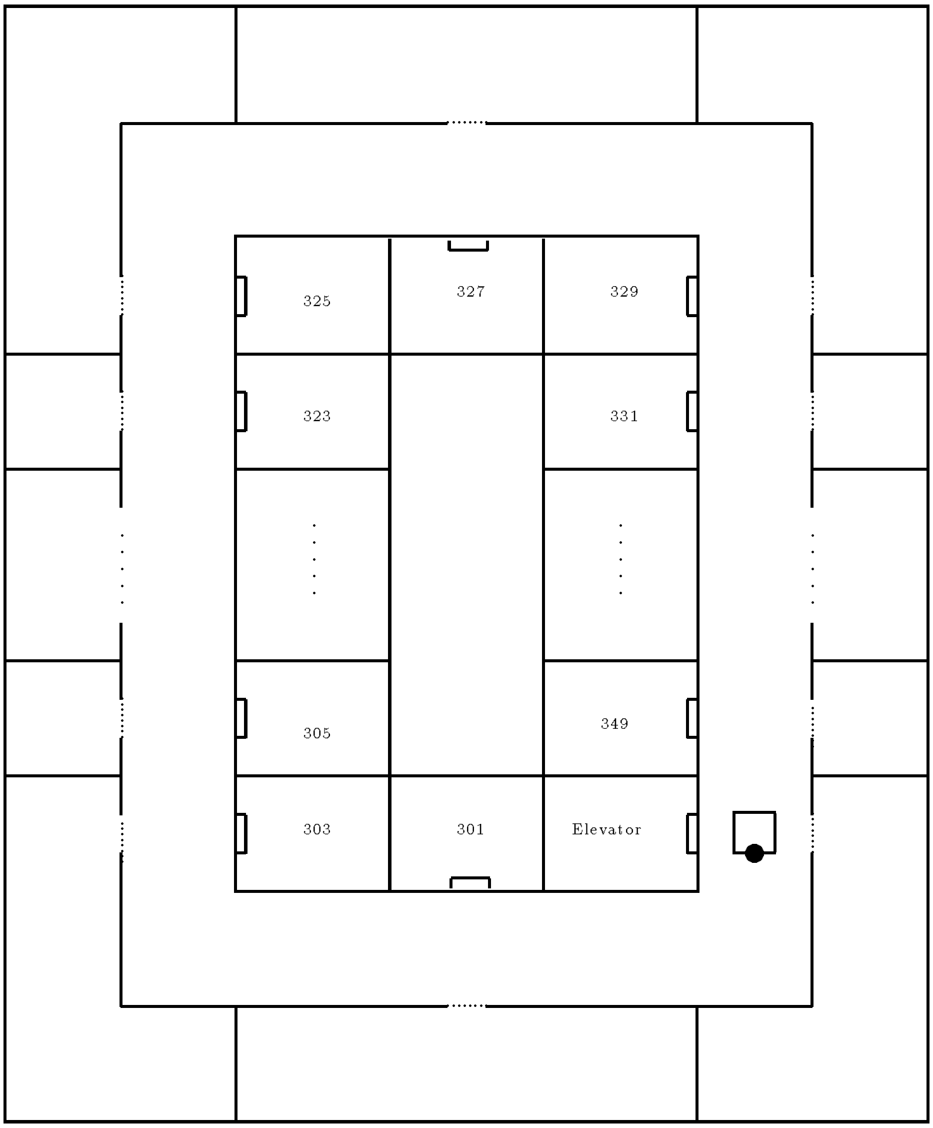
- *Control module* is a collection of control rules.
- Achievement control modules, and mixed control modules consist of only simple and termination control rules.
- Maintenance control modules consist of only simple and suspension control rules.

Operational semantics of simple control modules

- A simple control module can be in four different states: *active, suspended, success-terminated*, and *failure-terminated.*
- In the active state it continuously executes the following loop: *observe, match, and act.*
- In the observe cycle it reads its sensor values and quickly computes and updates the fluent values.
- In the match cycle it matches the values of the fluents with the LHS of the rules.
- In the act cycle it executes the actions in the RHS of all the rules whose LHS was matched successfully.

  If there are more than one such rules and the actions in their RHS are different but non-contradicting then it executes them concurrently.
- If the RHS of the rule is $HALT$ then the control module reaches the *success-terminated* state.
- If the RHS of the rule is $SUSPEND$ then the control module reaches the *suspended* state.

- In the suspended state the sensors are active and any change in the sensor values takes the robot from the *suspended* state to the *active* state.
- If in the match cycle no rule is found whose LHS is matched then the control module reaches the *failure-terminated* state.



Mobile robot in an office floor

Some Examples of Control Modules

Module : Goto_elevator_1

if $\neg at\_elevator$ **then** $Go\_clockwise\_1room$
if $at\_elevator$ **then** $HALT$

Specifying effects of the actions in module Goto_elevator_1

$Go\_clockwise\_1room$ **causes** $at(X+2)$
if $at(X), \neg at(349)$
$Go\_clockwise\_1room$ **causes** $at\_elevator$ **if** $at(349)$
$Go\_clockwise\_1room$ **causes** $at(301)$ **if** $at\_elevator$

Specifying constraints about the world corresponding to the module Goto_elevator_1

always $(at(301) \oplus \ldots \oplus at(349) \oplus at\_elevator)$[1]

[1] The symbol $\oplus$ denotes ex-or.

Another Example

Module : Goto_elevator_2

if $at(X), X \geq 325$ **then** $Go\_clockwise\_1room$
if $at(X), X \leq 323$ **then** $Go\_anticlockwise\_1room$
if $at\_elevator$ **then** $HALT$ $\square$

Specifying the effect of the action Go_anticlockwise_1room

$Go\_anticlockwise\_1room$ **causes** $at(X-2)$
if $at(X), \neg at(301)$
$Go\_anticlockwise\_1room$ **causes** $at\_elevator$
if $at(301)$
$Go\_anticlockwise\_1room$ **causes** $at(349)$
if $at\_elevator$

Defining $Go\_clockwise\_1room$ as a control module

Assumptions about the environment

- Doors of the outer rooms are painted white.
- Doors of the inner rooms are painted black.
- The control program that takes the robot out of a room also aligns the robot such that its side faces the door.

  When this control module is called for execution the fluent $just\_started$ is assigned the value $false$.
- $just\_started$ is a fluent that does not correspond to any sensor reading but encodes the state of the robot.

Module : Go_clockwise_1room

if $white\_door\_on\_rt$ **then** $turn\_180$
if $just\_started, black\_door\_on\_rt$ **then** $go\_forward$
if $just\_started, wall\_on\_rt$ **then** $del\_started$
if $\neg just\_started, wall\_on\_rt$ **then** $go\_forward$
if $corridor\_on\_rt$ **then** $turn\_rt\_90$
if $\neg just\_started, black\_door\_on\_rt$ **then** $HALT$ $\square$

Action theory for actions in Go_clockwise_1room

$turn\_180$ **causes** $black\_door\_on\_rt$ **if** $white\_door\_on\_rt$

$go\_forward$ **causes** $wall\_on\_rt$ **if** $black\_door\_on\_rt$
$go\_forward$ **causes** $black\_door\_on\_rt$ **if** $wall\_on\_rt, \neg app\_corner$
$go\_forward$ **causes** $at(X+2)$ **if** $wall\_on\_rt, \neg app\_corner, at(X), \neg at(349)$
$go\_forward$ **causes** $at\_elevator$ **if** $wall\_on\_rt, \neg app\_corner, at(349)$
$go\_forward$ **causes** $at(301)$ **if** $wall\_on\_rt, \neg app\_corner, at\_elevator$
$go\_forward$ **causes** $corridor\_on\_rt$ **if** $wall\_on\_rt, app\_corner$
$go\_forward$ **causes** $\neg app\_corner$ **if** $wall\_on\_rt, app\_corner$
$turn\_rt\_90$ **causes** $wall\_on\_rt$ **if** $corridor\_on\_rt$
$del\_started$ **causes** $\neg just\_started$ **if** $wall\_on\_rt$
$go\_forward$ **causes** app_corner **if** $black\_door\_on\_rt, at(301)$
$go\_forward$ **causes** app_corner **if** $black\_door\_on\_rt, at(325)$
$go\_forward$ **causes** app_corner **if** $black\_door\_on\_rt, at(327)$
$go\_forward$ **causes** app_corner **if** $black\_door\_on\_rt, at\_elevator$

Relationship between fluents

always $(black\_door\_on\_rt \oplus white\_door\_on\_rt \oplus wall\_on\_rt \oplus corridor\_on\_rt)$

always $(at(other)$ iff $(at(303) \oplus \ldots \oplus at(323) \oplus at(329) \oplus \ldots \oplus at(349)))$

always $(at(301) \oplus \ldots \oplus at(349) \oplus at\_elevator)$

always $(app\_corner \Rightarrow \neg at(other))$

always $(corridor\_on\_rt \Rightarrow \neg at(other))$

always $(corridor\_on\_rt \Rightarrow \neg app\_corner)$

A maintainance control module

Module : Maintain_Siren

- **if** off_siren **then** $turn\_on\_siren$
- **if** $\neg off\_siren$ **then** $SUSPEND$

The effect of the action $turn\_on\_siren$ is specified as follows:

$turn\_on\_siren$ **causes** $\neg off\_siren$

Characterizing correctness: Intuition

- Control modules behaves according to *specifications.*
- Specification for achievement modules:
  When the *module* executed in a state (s) from a *given set of states (S)*, it always terminates (according to the *action theory*) in states where the *achievement goal* is satisfied.
- The sequence of actions obtained by *unfolding* the module w.r.t. the state s when executed in s reaches a state where the *goal* is satisfied.
- *But what about exogenous actions?*
- Incorporated by expanding S.

Characterizing correctness: Closure

Definition 1 Let S be a set of states, M be a control module and A be an action theory. We say a set of states S' is a closure of S w.r.t. M and A, if S' is a minimal (w.r.t. subset ordering) set of states that satisfies the following conditions:

(i) $S \subseteq S'$.

(ii) If $s \in S'$ and a is an action in A that can occur independent of the robot then $Res(a, s) \in S'$

(iii) If $s \in S'$ and there exist a rule in M whose LHS is satisfied by s and whose RHS is $a_1, \ldots, a_l$, then $[a_1, \ldots, a_l]s \in S'$ □

Proposition 2 For any set of states S, control module M and action theory A, there is a unique set of states S' which is the closure of S w.r.t. M and A. □

Definition 3 A set of states S is said to be *closed* w.r.t. a control module M and an action theory A if $S = Closure(S, M, A)$. □

Formalizing Correctness

- *Unfolding of a control module w.r.t. to a state*
Intuitively, given a state s, and a simple control module M, the unfolding of M w.r.t. s ($\mathcal{U}_M(s)$) is the sequence (possibly infinite) of actions that the control module will execute starting from s, *in the absence of any exogenous actions.*

Definition 4 An achievement control module M is said to achieve goal G from a set of states S and w.r.t an action theory A, if for all s in $Closure(S, M, A)$, $\mathcal{U}_M(s)$ is finite and does not end with a_F^M and for all f in G,
$\models holds(f, [\mathcal{U}_M(s)]s)$.

Furthermore, M is said to n-achieve goal G from S, if
$max_{s \in S}|\mathcal{U}_M(s)| = n$. $\square$

Correctness: further extensions

- G could be a classical formula or even a temporal formula.
- Maintainance control modules
  Maintaining G is not same as keeping G true in all states. It means:
  - The agent won't make it false by its own actions;
  - If it is made false by exogenous actions, then the agent will act towards making it true; and
  - Given a sufficient window of non-interference from exogenous actions, the agent will make the goal true.

Sufficiency Conditions for correctness

- *Motivation:*
  - May make verification easier.
  - Useful in automatic construction.
  - Gives meaning to individually learnt rules.
- *Completeness:*
  An achievement control module M is said to be *complete* w.r.t. goal G and a set of states S, if for each s in S there exists at least one rule in M which is applicable in s.
- *Soundness:*
  The soundness condition can be intuitively stated (using the notion of action leading to a goal by Kaelbling and Rosenschein) as that a simple control rule r is sound if for any state where r is applicable the *action* in the RHS of r '*leads to the goal.*'

Sufficiency Condition: soundness

- An action *a weakly leads* to a goal from a state s if there exists a minimal plan from s which achieves the goal and has a as its first action.
- An action *a leads to a goal* from a state s if there exists a plan with minimal cost from s which achieves the goal and has a as its first action, where actions have an associated cost (a positive integer) and the cost of a plan is the sum of the cost of each of the actions in that plan.

Sufficiency: Weakly leading is not adequate!

Module : Goto_elevator_3

if $at(X), X \geq 325$ **then** $Go\_clockwise\_1room$
if $at(323),$ **then** $Go\_anticlockwise\_1room$
if $at(321),$ **then** $Go\_clockwise\_1room$
if $at(X), X \leq 319$ **then** $Go\_anticlockwise\_1room$
if $at\_elevator$ **then** $HALT$ □

- The above module is complete.
- For each possible state, there is a unique rule in the above module that is applicable to that state, and the RHS of that rule is an action that 'weakly leads' to the goal.
- But, if the robot is at room 323 or at room 321, and it executes the module $Goto\_elevator\_3$, it gets stuck in a loop.

Soundness: formal definition

Definition 5 (i)A simple control rule r is said to be *sound* w.r.t. a goal G and a set of states S (or w.r.t. (G, S)) if for all $s \in S$ such that r is applicable in s there exists a **minimal cost plan** that achieves G from s and has the RHS of r as its prefix.

(ii) A termination control rule r is said to be *sound* w.r.t. goal G and a set of states S (or w.r.t. (G, S)) if its LHS satisfies G.

An achievement control module M is *sound* w.r.t. goal G and a set of states S (or w.r.t. (G, S)) if each rule $r \in M$ is sound w.r.t (G, S). $\Box$

Sufficiency conditions guarantee correctness

Theorem 6 Let A be an action theory. Consider a pair (M, S), where S is a set of complete states, and M is a simple control module sequential w.r.t. S (one rule for each state in S) and S is closed w.r.t. M and A. Given a goal G, if M is sound and complete w.r.t. G and S then M achieves G from S w.r.t. A. □

- $Goto\_elevator\_1$ is sound and complete w.r.t. the set of states $\{\{at\_elevator\}, \emptyset\}$, when the action theory only consists of the action $Go\_clockwise\_1room$.

  When we add the action $Go\_anticlockwise\_1room$, it is no longer sound, but only weakly sound. We can no longer use Theorem 6 to verify its correctness.
- The control module $Goto\_elevator\_2$ is sound and complete w.r.t. the set of states $\{\{at\_elevator\}, \emptyset\}$, when the action theory consists of actions $Go\_clockwise\_1room$ and $Go\_anticlockwise\_1room$.
- The control module $Go\_clockwise\_1room$ is sound and complete w.r.t. the set of all states $\{\{at\_elevator\}, \emptyset\}$, and the action theory given earlier.

- The soundness and completeness conditions and Theorem 6 can be modified in an intuitive manner for mixed and maintainance control modules.
- There now exist incremental planners which can determine the prefix of plans, without constructing the whole plan.

Automatic construction of achievement modules

Input: Goal G, a set of states S, an action theory A for the robot and for the exogenous events.

Output: A Control Module M that achieves G from $Closure(S, M, A)$.

Step 1: $M = \{$ **if** G **then** $HALT\}$,
$S_{in} = \{s' : s'$ is reachable from some state s in S by applying sequences of exogenous actions$\}$ and $S_{out} = \emptyset$.

Step 2: While $S_{in} \neq S_{out}$
2.1 Pick a state s from $S_{in} \setminus S_{out}$
2.2 Find a minimal cost plan P from s that achieves G. Let a be the first action in P.
2.3 If P is a not a null plan then
$M = M \cup \{$ **if** s **then** $a\}$;
2.4 $S_{in} = S_{in} \cup [a]s \cup \{s' : s'$ is reachable from $[a]s$ by

applying sequences of exogenous actions} and
$S_{out} = S_{out} \cup \{s\}$.

Step 3: Merge control rules (using circuit optimization techniques) in M which have the same RHS and whose LHS can be combined.

Proposition
Given a goal G, a set of states S, and an action theory A if G is achievable from all states in $Closure(S, A, A)$, then the above algorithm generates a control module M such that M achieves G from $Closure(S, M, A)$.

An algorithm using weakly leading condition

Step 1 $M = \{$ **if** G **then** $HALT\}$,
$S_{in} = S \cup \{s' : s'$ is reachable from some state s in S by applying sequences of exogenous actions$\}$ and $S_{out} = \emptyset$.

Step 2 While $S_{in} \neq S_{out}$
2.1 Pick a state s from $S_{in} \setminus S_{out}$
2.2 If s satisfies G then $S_{out} = S_{out} \cup \{s\}$ and go to step 2. Otherwise, go to step 2.3
2.3 Find the minimal cost plan P from s that achieves G. Let the minimal plan be $a_1, \ldots, a_k$.
2.4 Let i be the smallest number such that $[a_1, \ldots, a_i]s \in S_{out}$. (If $[a_1, \ldots, a_i]s \notin S_{out}$ for $i = 1, \ldots, k$ then we set $i = k + 1$.)
2.5 Let
$M = M \cup \{$ **if** $[a_1, \ldots, a_j]s$ **then** $a_{j+1} : 0 \leq j < i\}$;
2.6 $S_{in} = S_{in} \cup \{[a_1, \ldots, a_j]s \ : \ 0 \leq j < i\} \cup \{s' : s'$ is reachable from $[a_1, \ldots, a_j]s$ for some $j < i$ by applying sequences of exogenous actions$\}$ and
$S_{out} = S_{out} \cup \{[a_1, \ldots, a_j]s : 0 \leq j < i\}$.

Step 3 Merge control rules in M which have the same RHS and whose LHS can be combined.

Proposition:
Given a goal G, a set of states S and an action theory A if G is achievable from all states in $Closure(S, A, A)$, then the above algorithm generates a control module M such that M achieves G from $Closure(S, M, A)$.

CONTROL MODULE WITH SENSING ACTIONS

A control module for getting milk

Example 7 Control Module – Get_Milk

if $\neg has\_milk,\ u(fridge\_open), \neg near\_fridge$
then go_to_fridge

if $\neg has\_milk,\ u(fridge\_open), near\_fridge$
then $look\_at\_fridge$

if $\neg has\_milk,\ \neg fridge\_open, \neg near\_fridge$
then go_to_fridge

if $\neg has\_milk,\ \neg fridge\_open, near\_fridge$
then $open\_fridge$

if $\neg has\_milk,\ fridge\_open,\ u(milk\_in\_fridge),$
$near\_fridge$ **then** $look\_into\_fridge$

if $\neg has\_milk,\ fridge\_open,\ u(milk\_in\_fridge),$
$\neg near\_fridge$ **then** go_to_fridge

if $\neg has\_milk,\ milk\_in\_fridge,\ fridge\_open, \neg near\_fridge$
then go_to_fridge

if $\neg has\_milk,\ milk\_in\_fridge,\ fridge\_open, near\_fridge$
then $get\_milk\_from\_fridge$

if $\neg has\_milk,\ \neg milk\_in\_fridge$ **then** $get\_milk\_from\_store$

if has_milk **then** $HALT$ □

Causal Rules for actions in Get_Milk

go_to_fridge **causes** $near\_fridge$
$look\_at\_fridge$ **determines** $fridge\_open$ **if** $near\_fridge$
$open\_fridge$ **causes** $fridge\_open$ **if** $near\_fridge$
$look\_into\_fridge$ **determines** $milk\_in\_fridege$
if $fridge\_open, near\_fridge$
$get\_milk\_from\_fridge$ **causes** has_milk
if $milke\_in\_fridge, fridge\_open, near\_fridge$
$get\_milk\_from\_store$ **causes** has_milk

Action theory with sensing actions

- Scherl and Levesque (AAAI 93) – States are Kripke models.
- Lobo, Taylor and Mendez (AAAI 97) – States are sets of 3-valued interpretations.
- Baral and Son (ILPS 97) – Approximate semantics. States are 3-valued interpretations.
- POMDP

Unfolding: results in conditional plans

Conditional Plan - $Plan\_Get\_Milk$

$go\_to\_fridge;$
$look\_at\_fridge;$
Case
$\neg fridge\_open \rightarrow open\_fridge;$
  $look\_into\_fridge;$
  Case
  $milk\_in\_fridge \rightarrow get\_milk\_from\_fridge$
  $\neg milk\_in\_fridge \rightarrow get\_milk\_from\_store$
  Endcase
$fridge\_open \rightarrow look\_into\_fridge;$
  Case
  $milk\_in\_fridge \rightarrow get\_milk\_from\_fridge$
  $\neg milk\_in\_fridge \rightarrow get\_milk\_from\_store$
  Endcase
Endcase

Correctness, sufficiency conditions

- A sensing action is a mapping from a state to a set of states; but differs from actions with non-deterministic effects.
- Correctness: Similar to the earlier definition.
- Sufficiency conditions:
  - Need to define minimal cost conditional plans
  - Need to consider knowledge preserving actions and plans.
  - Instead of arbitrary prefix of minimal cost plans, need to consider a prefix that has at least one sensing action.
  - Automatic construction algorithm is very similar.

FROM THEORY TO PRACTICE

AAAI 96 and AAAI 97 robot contests

- AAAI 96:
  3rd in the office navigation contest.
  285 out of 295 possible points.
- AAAI 97:
  1st in the home vacuuming contest.

AAAI 96: The top level control module

if $\neg visit\_conf\_1$ **then** $go\_to\_conf(1)$
if $at\_conf(1), u(avail(1))$ **then** $sense\_avail(1)$
if $at\_conf(1), \neg avail(1)$ **then** $go\_to\_conf(2)$
if $at\_conf(1), avail(1), \neg visit\_prof(1)$ **then** $go\_to\_prof(1)$
if $at\_conf(2), u(avail(2))$ **then** $sense\_avail(2)$
if $at\_conf(2), avail(2), \neg visit\_prof(1)$ **then** $go\_to\_prof(1)$
if $at\_conf(2), \neg avail(2), \neg visit\_prof(1)$ **then** $go\_to\_prof(1$
if $at\_prof(1), \neg visit\_prof(2)$ **then** $go\_to\_prof(2)$
if $at\_prof(2), \neg back\_to\_director$ **then** $go\_to\_director$
if $back\_to\_director$ **then** $HALT$

AAAI 96: Specifying the goal in Temporal logic

- Ka means a is known to be true;
- avail(1) means conference room 1 is available,
- informedprof1(1) means professor 1 has been informed that the meeting will be in conference room 1,
- $\Box f$ means always f is true,
- $\Diamond f$ means eventually f is true, and
- $at(dir)$ means the robot is at the directors office.

The Goal:

$$
\begin{array}{l}
(Kavail(1) \vee K\neg avail(1)) \wedge \\
(K\neg avail(1) \Rightarrow (Kavail(2) \vee K\neg avail(2))) \wedge \\
(Kavail(1) \Rightarrow (informedprof1(1) \wedge \\
informedprof2(1) \wedge informeddirector(1))) \wedge \\
((Kavail(2) \wedge K\neg avail(1)) \Rightarrow (informedprof1(2) \wedge \\
informedprof2(2) \wedge informeddirector(2))) \wedge \\
((K\neg avail(1) \wedge K\neg avail(2)) \Rightarrow (informedprof1(dir) \wedge \\
informedprof2(dir) \wedge informeddirector(dir))) \wedge \\
\forall X avail(X) \Rightarrow \Box avail(X) \wedge \\
\Box clear\_from\_obstacle \wedge \Diamond \Box at(dir)
\end{array}
$$

RELATED WORK

Universal plans and situation control rules

- Universal plans prescribe what actions need to be executed in *each possible situation.*
  In contrast our control modules *only consider the closure of a given set of initial states.*
- The sufficiency conditions about our control modules *guarantee* the correctness of the control module – thus strengthening the results by Schoppers and Drummond. In particular:
  - Schoppers in 1987 IJCAI says, "Universal plans not only anticipate every possible situations in a domain but actually prescribe an action for every initial state; more over the prescribed action is *usually optimal.*"
    In our formulation the prescribed action is optimal.
  - Drummond in KR89 "sound SCRs guarantee that local execution choices always lead to *possible goal achievement*".
    In our formulation, local execution choices always lead to goal achievement.

- Drummond et al. in AAAI 90 present an algorithm for incremental control rule synthesis.
  Drummond et al. in AAAI 94 present an algorithm for building robust schedules that takes a nominal schedule and builds contingent schedules for anticipated errors.
  But, *they do not have a formal correctness result about their algorithms.*
- Kabanza et al. – Use temporal goals, action effects and environment are put together, constuct control module as a whole.
- Dean, Kaelbling et al. – probabilistic effects, action effects and environment are put together, policy iteration used in automatic construction of the control module as a whole.
- Use of shortest plans by Jonsson and Backstorm (AAAI 96) in constructing universal plans in the proof of Theorem 10 is similar to the minimal cost condition in our definition of soundness of control rules.
- Kaelbling and Rosenschein : 'Action leading to a goal'.
- GAPPS: Compilation based on HTN planning.
- Saffioti et al (AIJ 95) formalize the notion of correctness of control modules which use multi-valued logic to take into account uncertainty of the environment and sensor

noise.
But, they do not consider exogenous actions, and more important, they do not give any sufficiency conditions for correctness, nor any procedure to automatically construct control modules.

Additional Contributions

- We do not use any specific action theory.
- Formal treatment of 'maintainance goals' in presence of exogenous actions. (Most formulation of universal plans only consider achievement goals.)
- We consider special sensing actions. To our knowledge, no earlier formulation of universal plans or situation control modules (except, POMDPs) consider such actions.

Relation with agent theories and architectures

- Reactive agents:
  Our formulation of correctness can be easily adapted to the other related variants, such as the subsumption architecture of Brooks.
  We present algorithms to automatically construct control modules, and allow special sensing actions in our control modules, not considered in the earlier approaches.
- Hybrid Agents:
  Our approach is hybrid in a different sense – reactivity for anticipated highly plausible situations (situations in the closure set), while occasional deliberativity for rare situations that have not been taken into account during compilation.
- Formulation of agent behavior using the temporal logic CTL* :
  Worlds are represented as time trees.
- BDI agents:
  - *Beliefs* correspond to the action theory and the current values of the fluent.
  - *Desires* correspond to the goal.

- *We believe that intention in a particular situation corresponds to the plan obtained by unfolding the control module w.r.t. that state.*
  * *Immediate intention*
  * *n-action intention*
  * *n-action intention w.r.t. the exogenous action theory* A
  * In presence of sensing actions, an *n-action intention* at a situation, could be a conditional plan.

• Relation with GOLOG:

- Both formalize a notion of correctness of complex robot execution programs.
- GOLOG allows non-deterministic actions, and recursive procedures while we do not.
- GOLOG does not consider exogenous actions, and is not suitable for an agent in a dynamic environment, where changes beyond the control of the robot may occur.
- GOLOG, CONGOLOG and $\mathcal{R}$ are not concerned with automatic construction of robot programs, which is one of our main concerns.

• Reactive Architectures with on-line planning

- Kowalski
- Golden et al.

- Using action theories and reactive rules:
  - Wagner et al.
  - Li and Pereira

  Both do not consider correctness aspect of the reactive part, do not allow sensing actions, and do not consider automatic generation of the reactive part.

FUTURE DIRECTIONS

Future Directions

- Hierarchical control modules as a reactive robot execution language.
- Goals in LHS of control rules.
- Recognizing failure and recovery.
  (Random failure can be tackled, but systematic failure can not be.)
- Complex (temporal) Goals.
  (Correctness is straight forward. Automatic generation of control modules is not.)
- Co-operative control with multiple agents.
- Correctness and automatic construction of fuzzy control modules.
- Relation with MDPs and POMDPs.
- Implementation Issues.

CONCLUSION

Dijkstra's quote in 'Humble Programmer'

The only effective way to raise the confidence level of a program significantly is to give proof of its correctness. But one should not first make the program and then prove its correctness, because then the requirement of providing the proof will only increase the poor programmer's burden. On the contrary: the programmer should let correctness proof and program grow hand in hand. ... If one first asks oneself what the structure of a convincing proof would be, and having found this, then constructs a program satisfying this proof's requirements, then these correctness concerns turnout to be a very effective heuristic guidance.

$\boxed{\text{Easier to specify control than action theory.}}$

- $Go\_clockwise\_1room$ is perhaps such a module.
- But, there is a point in having alternate representations.
- Hoare in IEEE Computer 1987 says:

> For specifications of the highest quality and importance I would recommend complete formalization of requirements in two entirely different styles, together with a proof that they are consistent or even equivalent to each other. ... A language for which two consistent and complementary definitions are provided may be confidently taken as a secure basis for software engineering.

Conclusion: Scruffy vs formal reasoning

- In so called, 'scruffy methods' (situated automata, production systems, case based reasoning, frames, some expert systems), programs are developed that reason efficiently but the formal foundations are not clear.
- On the other hand in 'formal reasoning' (non-monotonic reasoning systems) the formal foundations are clear, but the reasoning is usually inefficient.
- The relation between the two is that the programs that do 'scruffy reasoning' are compiled (and hence are more efficient) from knowledge represented in a formal system, based on what is expected from the programs.
- Sufficient progress in both is needed to understand the compilation process.
- In this paper the 'programs that do scruffy reasoning' corresponds to the 'control modules', the 'knowledge represented in a formal system' corresponds to the 'action theory' and the 'expectation' corresponds to the expectation that the robot can act quickly after sensing.
- We believe this generalization will clear misunderstandings between many of the researchers following either of the approaches, and will increase respect towards each other.

- Some expert systems can be though of as 'Reactive abduction'.
- Stratified logic programs may be thought of as 'Reactive Deduction'.