Outline for 4/23

- Logistics
- Planning
- Review of last week
- Incorporating Dynamics
- Model-Based Reactive Planning

Logistics

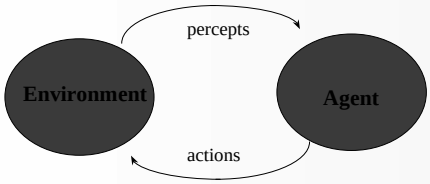
- Problem Set 2
- Project

Course Topics by Week

- Search & Constraint Satisfaction
- Knowledge Representation 1: Propositional Logic
- Autonomous Spacecraft 1: Configuration Mgmt
- Autonomous Spacecraft 2: Reactive Planning
- Information Integration 1: Knowledge Representation
- Information Integration 2: Planning & Execution
- Supervised Learning & Datamining
- Reinforcement Learning
- Bayes Nets: Inference & Learning
- Review & Future Forecast

Agent *vs.* Environment

- Agent has sensors, effectors
- Implements mapping from percept sequence to actions



Two Approaches to Agent Control

- ***Reactive Control***
  - Set of situation-action rules
  - E.g.
    - 1) **if** dog-is-behind-me
      then run-forward
    - 2) **if** food-is-near
      then eat
- ***Planning***
  - Reason about effect of combinations of actions
  - "Planning ahead"
  - Avoiding "painting oneself into a corner"

5

Planner

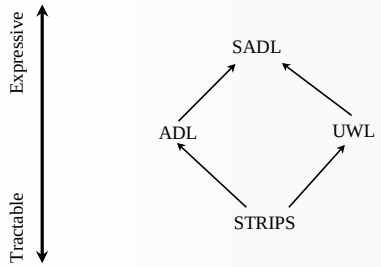
- Input
  - Description of initial state of world *(in some KR)*
  - Description of goal *(in some KR)*
  - Description of available actions *(in some KR)*
- Output
  - Sequence of actions

6

Input Representation

- *Description of initial state of world*
  - Set of propositions:
  - **((block a) (block b) (block c) (on-table a) (on-table b) (clear a) (clear b) (clear c) (arm-empty))**
- *Description of goal (i.e. set of desired worlds)*
  - Logical conjunction
  - Any world that satisfies the conjunction is a goal
  - **(:and (on a b) (on b c)))**
- *Description of available actions*

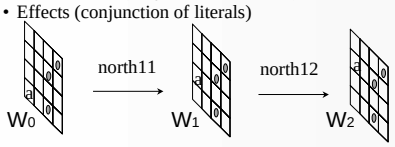
Representing Actions



8

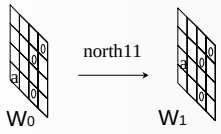
How Represent Actions?

- ***Simplifying assumptions***
  - Atomic time
  - Agent is omniscient (no sensing necessary).
  - Agent is sole cause of change
  - Actions have deterministic effects
- ***STRIPS representation***
  - World = set of true propositions
  - Actions:
    - Precondition: (conjunction of literals)
    - Effects (conjunction of literals)



STRIPS Actions

- Action = a function from world-state to world-state
- Precondition says when function defined
- Effects say how to change set of propositions



north11
precond: (:and (agent-at 1 1)
(agent-facing north))
effect: (:and (agent-at 1 2)
(:not (agent-at$_0$1 1)))

Action Schemata

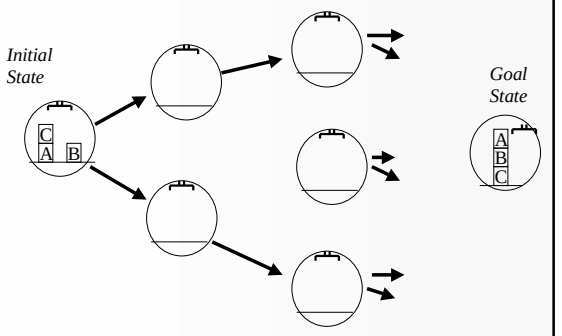
- Instead of defining: **pickup-A** and **pickup-B** and …
- Define a schema:

(:operator **pick-up**
:parameters ((block ?ob1))
:precondition (:and (clear ?ob1) (on-table ?ob1) (arm-empty))
:effect (:and (:not (on-table ?ob1))
(:not (clear ?ob1))
(:not (arm-empty))
(holding ?ob1)))

Planning as Search

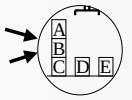
- Nodes
  World states
- Arcs
  Actions
- Initial State
  The state satisfying the complete description of the initial conds
- Goal State
  Any state satisfying the goal propositions

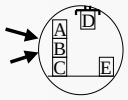
Forward-Chaining World-Space Search



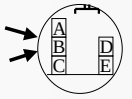
Backward-Chaining World-Space Search

- Problem: Many possible goal states are equally acceptable.
- From which one does one search?

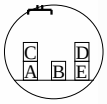




* * *



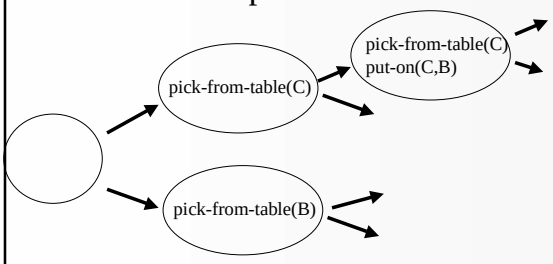
Initial State is completely defined



Planning as Search 2

- Nodes
  - Partially specified plans
- Arcs
  - Adding + deleting actions or constraints (e.g. <) to plan
- Initial State
  - The empty plan
- Goal State
  - A plan which when simulated achieves the goal

Plan-Space Search

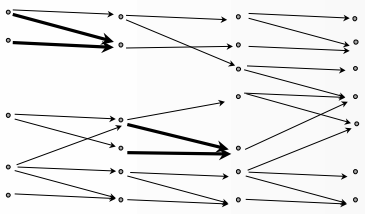


- How represent plans?
- How test if plan is a solution?

Planning as Search 3

- Phase 1 - Graph Expansion
  - **Necessary (insufficient) conditions for plan existence**
  - **Local consistency of plan-as-CSP**
- Phase 2 - Solution Extraction
  - **Variables**
    - **action execution at a time point**
  - **Constraints**
    - **goals, subgoals achieved**
    - **no side-effects between actions**

Planning Graph



18

Constructing the planning graph…

- Initial proposition layer
  - Just the initial conditions
- Action layer i
  - If all of an action's preconds are in i-1
  - Then add action to layer I
- Proposition layer i+1
  - For each action at layer i
  - Add all its effects at layer i+1

Mutual Exclusion

- Actions A,B ***exclusive (at a level)*** if
  - A deletes B's precond, or
  - B deletes A's precond, or
  - A & B have inconsistent preconds
- Propositions P,Q ***inconsistent (at a level)*** if
  - all ways to achive P exclude all ways to achieve Q

20

Graphplan

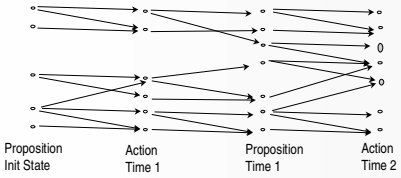
- Create level 0 in planning graph
- Loop
  - If goal $\subseteq$ contents of highest level (nonmutex)
  - Then search graph for solution
    - If find a solution then return and terminate
  - Else Extend graph one more level

A kind of double search: forward direction checks necessary (but insufficient) conditions for a solution, ... Backward search verifies...

21

Searching for a Solution

- For each goal G at time t
  - For each action A making G true @t
    - If A isn't mutex with a previously chosen action, select it
    - If no actions work, backup to last G (breadth first search)
- Recurse on preconditions of actions selected, t-1



22

Dinner Date

Initial Conditions: (:and (cleanHands) (quiet))

Goal: (:and (noGarbage) (dinner) (present))

Actions:

(:operator **carry** *:precondition*
:effect (:and (noGarbage) (:not (cleanHands)))
(:operator **dolly** *:precondition*
:effect (:and (noGarbage) (:not (quiet)))
(:operator **cook** *:precondition* (cleanHands)
:effect (dinner))
(:operator **wrap** *:precondition* (quiet)
:effect (present))

Planning Graph

noGarb

carry

cleanH cleanH

dolly

quiet quiet

cook

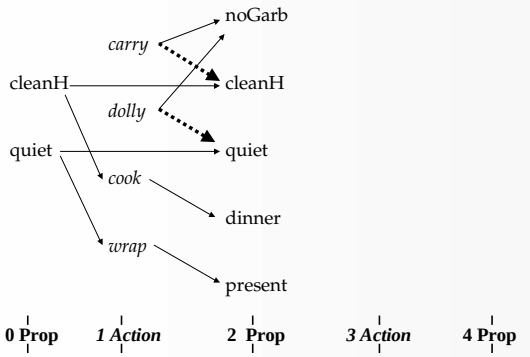
dinner

wrap

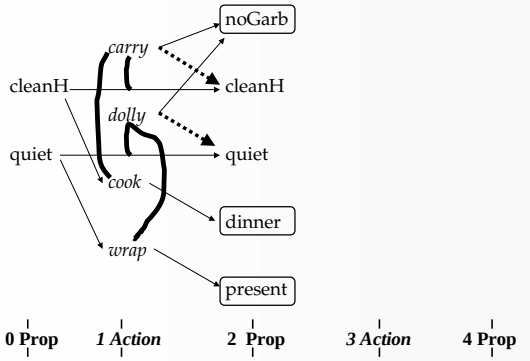
present

0 Prop ***1 Action*** **2 Prop** ***3 Action*** **4 Prop**

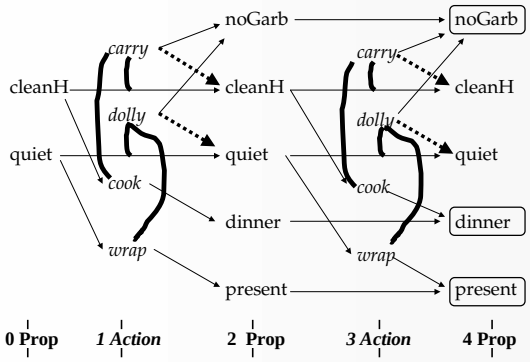
Are there any exclusions?



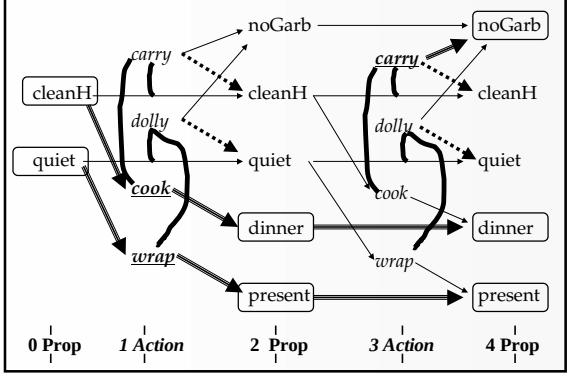
Do we have a solution?



Extend the Planning Graph



One (of 4) possibilities



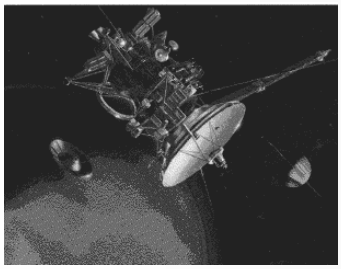
Summary Planning

- Reactive systems *vs.* planning
- Planners can handle medium to large-sized problems
- Relaxing assumptions
  - Atomic time
  - Agent is omniscient (no sensing necessary).
  - Agent is sole cause of change
  - Actions have deterministic effects
- Generating contingent plans
  - Large time-scale Spacecraft control

Outline

- Logistics
- Planning
- Review of last week
- Incorporating Dynamics
- Model-Based Reactive Planning

Immobile Robots Example 2: Cassini Saturn Mission



- ~ 1 billion $
- 7 years to build
- 7 year cruise
- ~ 150 - 300 ground operators

⇩

- *150 million $*
- *2 year build*
- *0 ground ops*

Solution: Part 1 Model-based Programming

Programmers and operators generate breadth of functions from commonsense hardware models in light of mission-level goals.

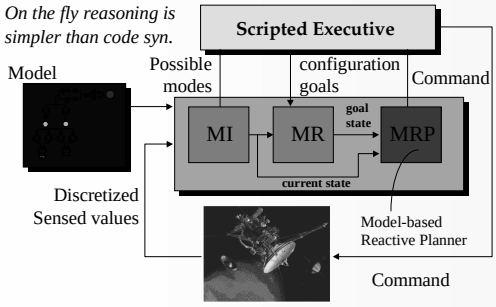
⇩

Have engineers program in models, automate synthesis of code:

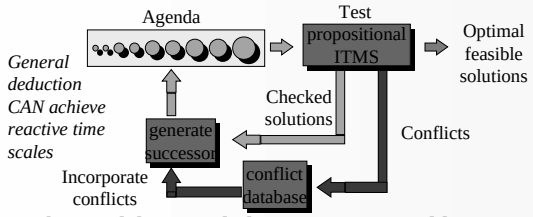
- models are compositional & highly reusable.
- generative approach covers broad set of behaviors.
- commonsense models are easy to articulate at concept stage and insensitive to design variations.

Solution: Part 2 Model-based Deductive Executive

On the fly reasoning is simpler than code syn.

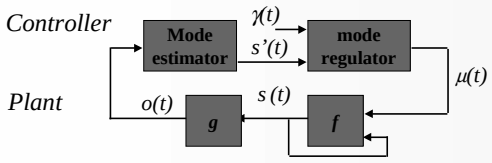


Solution: Part 3
Risc-like Best-first, Deductive Kernel



- Tasks, models compiled into propositional logic
- Conflicts dramatically focus search
- Careful enumeration grows agenda linearly
- ITMS efficiently tracks changes in truth assignments

Consider a sub family of model-based optimal controllers where...



$\mu = argmin\ C(s', \gamma, \mu')\ s.t.\ \mu' \in M$

- $s(t)$, $s'(t)$, $o(t)$, $\gamma(t)$, $\mu(t)$ have discrete, finite domains and time t is discrete.
- f and g are specified declaratively.
- the estimator and regulator are implemented as queries to a fast, best first, propositional inference kernel.

A family of increasingly powerful deductive model-based optimal controllers

- Step 1: Model-based configuration management with a partially observable state-free plant.
- Step 2: Model-based configuration management with a dynamic, concurrent plant.
- Step 3: Model-based executive with a reactive planner, and an indirectly controllable dynamic, concurrent plant.

Specifying a valve

- Variables $\Pi = \{mode, f_{in}, f_{out}, p_{in}, p_{out}\}$
  - $mode \in \{open, closed, stuck\text{-}open, stuck\text{-}closed\}$
  - f_{in}, and f_{out} range over $\{positive, negative, zero\}$
  - p_{in}, and p_{out} range over $\{high, low, nominal\}$
- Specifying Σ with ρ_Σ

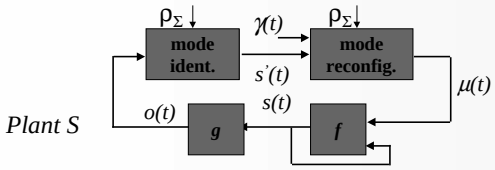
  $mode = open \Rightarrow (p_{in} = p_{out}) \wedge (f_{in} = f_{out})$

  $mode = closed \Rightarrow (f_{in} = zero) \wedge (f_{out} = zero)$

  $mode = stuck\text{-}open \Rightarrow (p_{in} = p_{out}) \wedge (f_{in} = f_{out})$

  $mode = stuck\text{-}closed \Rightarrow (f_{in} = zero) \wedge (f_{out} = zero)$

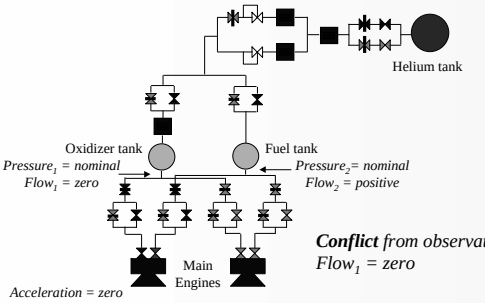
Mode identification + reconfiguration



Configuration management achieved by

- Mode identification
  - identifies the system state based only on observables
- Mode reconfiguration
  - reconfigures the system state to achieve goals

Example: Cassini propulsion system



Conflict *from observation*
$Flow_1 = zero$

MI/MR as combinatorial optimization

- MI
  - variables: components with domains the possible modes
    - an assignment corresponds to a candidate diagnosis
  - feasibility: consistency with observations
  - cost: probability of a candidate diagnosis
- MR
  - variables: components with domains the possible modes
    - an assignment corresponds to a candidate repair
  - feasibility: entailment of goal
  - cost: cost of repair

Generic LTMS interface

- Updating the clauses in Σ
  - *add-clause* (*clause,* Σ)
  - *delete-clause* (*clause,* Σ)
- Propositional inference
  - *consistent?* (Σ)
  - *follows-from?* (*literal,* Σ)
- Justification structure
  - *supporting-clause* (*literal,* Σ)
  - *supporting-literals* (*literal,* Σ)
    - the *supporting-clause* together with the *supporting-literals* entail *literal*
    - each literal in *supporting-literals* follows from Σ

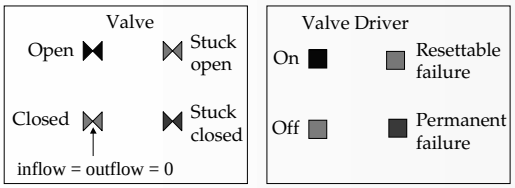
Using the LTMS in MI and MR

- LTMS database Σ contains clauses describing component behavior in each mode
- MI searches for component mode assignments that are consistent with the observations
- MR searches for component mode assignments that entail the goal
- Both add + delete clauses corresponding to assumptions that a component is in particular mode
- Justification structure is used to generate conflicts from an inconsistent Σ

Outline

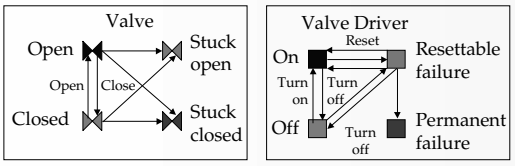
- Logistics
- Planning
- Review of last week
- Incorporating Dynamics
- Model-Based Reactive Planning

Plain Propositional Logic



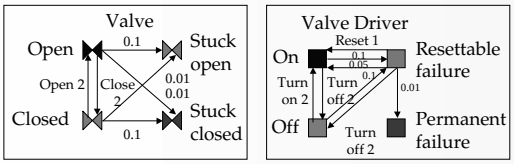
- Models behavior only *within* a component mode
- Does not explicitly model *transitions* between modes

Transition systems



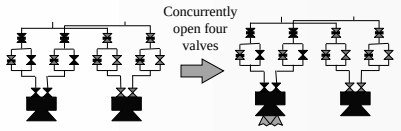
- Explicitly model mode transitions (including self-transitions)
  - commanded transitions with preconditions
  - failure (uncommanded) transitions
  - repair transitions
  - intermittency

Markov models



- Represent probability of uncommanded transitions and cost of commanded transitions
- Can model
  - Reliability
  - Optimal control

Concurrent transition systems

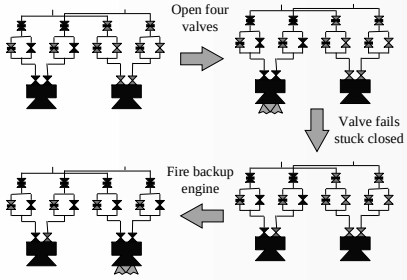


- Components within a system are modeled as *concurrent* transition system [Manna & Pneuli 92]

⇒Each system transition consists of a single transition by *each* component transition system (possibly the idling transition)

- Can naturally model digital hardware, analog hardware using qualitative representations, and real-time software

Trajectories of concurrent transition systems



Transition system models

- A system **S** is a tuple (Π, Σ, T)
- Π: set with *variables* ranging over finite domains
  - *state* variables (Π_s)
  - *control* variables (Π_c)
  - *dependent* variables (Π_d)
  - *observable* variables (Π_o)
- Σ : set of *feasible assignments*
  - Let Σ_s be the projection of Σ on Π_s
  - each element of Σ_s is a *state*
- T **: set of *transitions***
  - **each transition is a function $\Sigma \rightarrow \Sigma_s$**
  - **a single transition $\tau_n \in$ T is the *nominal* transition**
  - **all other transitions are *failure* transitions**

Specifying transition systems

- System S = (Π, Σ, T) is specified using a *propositional temporal logic* formula ρ_S
- Propositions are of the form $y_k = e_k$
  - y_k is a variable in Π, and e_k is in y_k's domain
- Feasible assignments spec. by a prop. formula ρ_Σ
  - E.g. Σ = the set of assignments that satisfy ρ_Σ
- **Each transition $\tau \in$ T specified using a conjunction ρ_τ of formulas of the form ρ_{τ_i} : $\Phi_{\tau_i} \Rightarrow$ *next* (Ψ_{τ_i})**
  - **Φ_{τ_i} is a propositional formula**
  - **Ψ_{τ_i} is of the form $y_k = e_k$ for state variable y_k**
  - **$\tau(a) = s$ if and only if for each ρ_{τ_i} whenever assignment *a* satisfies Φ_{τ_i} then *s* satisfies Ψ_{τ_i}**

Specifying a valve transition system

Same as for state-free systems:

- Variables Π = {*mode, cmd,* f_{in}, f_{out}, p_{in}, p_{out} }
  - *mode* $\in$ {*open, closed, stuck-open, stuck-closed*}
  - *cmd* ranges over {*open, close, no-cmd*}
  - f_{in}, and f_{out} range over {*positive, negative, zero*}
  - p_{in}, and p_{out} ranging over {*high, low, nominal*}
- Specifying Σ with ρ_Σ

  mode = open $\Rightarrow$ ($p_{in} = p_{out}$) Λ ($f_{in} = f_{out}$)

  mode = closed $\Rightarrow$ ($f_{in} = zero$) Λ ($f_{out} = zero$)

  mode = stuck-open $\Rightarrow$ ($p_{in} = p_{out}$) Λ ($f_{in} = f_{out}$)

  mode = stuck-closed $\Rightarrow$ ($f_{in} = zero$) Λ ($f_{out} = zero$)

Valve transition system (cont.)

- Specifying the nominal transition $\tau_n \in T$
  mode = closed Λ *cmd = open* $\Rightarrow$ *next* (*mode = open*)
  mode = closed Λ *cmd* $\neq$ *open* $\Rightarrow$ *next* (*mode = closed*)
  mode = open Λ *cmd = close* $\Rightarrow$ *next* (*mode = closed*)
  mode = open Λ *cmd* $\neq$ *close* $\Rightarrow$ *next* (*mode = open*)
  mode = stuck-open $\Rightarrow$ *next* (*mode = stuck-open*)
  mode = stuck-closed $\Rightarrow$ *next* (*mode = stuck-closed*)
- Specifying failure transitions
  - τ_1 : *mode = closed* $\Rightarrow$ *next* (*mode = stuck-closed*)
  - τ_2 : *mode = closed* $\Rightarrow$ *next* (*mode = stuck-open*)
  - τ_3 : *mode = open* $\Rightarrow$ *next* (*mode = stuck-open*)
  - τ_4 : *mode = open* $\Rightarrow$ *next* (*mode = stuck-closed*)

Configuration Manager

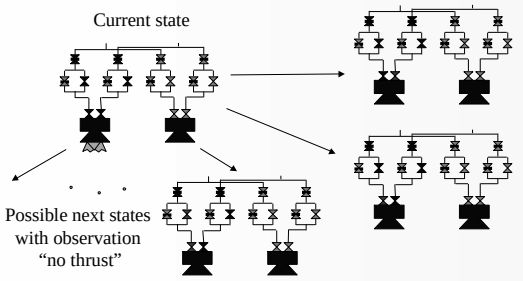
Input

- *S* is a system (Π, Σ, T)
- γ : g_0, g_1 … , called goal configurations, is a sequence of propositional formulae on Π
- *O* : o_0, o_1 … , is a sequence of observations

Output

- μ : μ_0, μ_1 …, a sequence of values for all control variables so that *S* evolves through a *configuration trajectory* σ : s_0, s_1 … such that
  - for every assignment a_i that agrees with s_i, o_i, and μ_i
  - **if $s_{i+1} = \tau_n(a_i)$, then s_{i+1} satisfies goal configuration g_i**

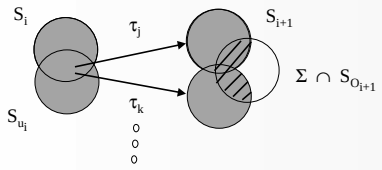
Mode Identification



Characterizing MI

- Find possible next states, given current state, commands, constraints and next state observations.

$$S_{i+1} = \left(\bigcup_j \tau_j (S_i \cap S_{\mu_i}) \right) \cap \Sigma \cap S_{O_{i+1}}$$



Characterizing MI

- Possible next states given current state commands and next state observations

$$S_{i+1} = \left(\bigcup_j \tau_j (S_i \cap S_{\mu_i}) \right) \cap \Sigma \cap S_{O_{i+1}}$$

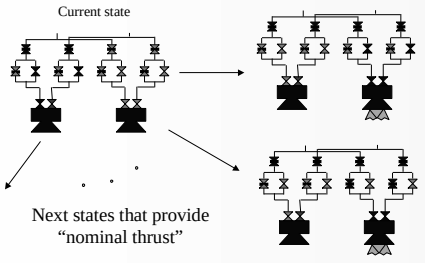
- Characterization of the possible next states

$$\rho_{S_{i+1}} \equiv \bigvee_{\tau_j} \left(\rho_{S_i} \wedge \rho_{S_{\mu_i}} \wedge \text{entails } \Phi_{jk}\ \Psi_{jk} \right) \wedge \rho_\Sigma \wedge \rho_{O_{i+1}}$$

where transition τ_j is specified by a conjunction of formulas $\Phi_{jk} \Rightarrow next(\Psi_{jk})$

- Focus on likely transitions

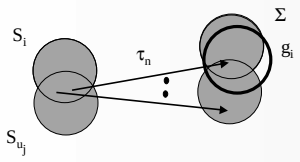
Mode Reconfiguration: Reachability in the next state



Characterizing MR

- Find possible commands that achieve the current goal in the next state.

$$\mathrm{M}_i = \left\{ \mu_j \mid \tau_n \left(S_i \cap S_{\mu_j} \right) \cap \Sigma \subseteq g_i \right\}$$



Characterizing MR

- Possible commands that achieve the current goal in the *next* state for all predicted trajectories

$$\mathrm{M}_i = \left\{ \mu_j \mid \tau_n \left(S_i \cap S_{\mu_j} \right) \cap \Sigma \subseteq g_i \right\}$$

- Characterization of the possible commands

$$\mathrm{M}_i = \left\{ \mu_j \mid \rho_{S_i} \wedge \rho_{\mu_j} \text{ is consistent and } \left(\bigwedge_{\rho_{S_i} \wedge \rho_{\mu_j} \text{ entails } \Phi_{ik}} \Psi_{ik} \right) \wedge \rho_\Sigma \text{ entails } \rho_{g_i} \right\}$$

- Focus on optimal control actions

Statistically Optimal Configuration Management

Statistical Mode Identification

- $p(\tau_j \mid o_i) = p(o_i \mid \tau_j)\, p(\tau_j) / p(o_i)$ Bayes Rule
  $\propto p(o_i \mid \tau_j)\, p(\tau_j)$
- $p(o_i \mid \tau_j)$ is approximated from the model
  - $p(o_i \mid \tau_j) = 1$ if $\tau_j(a_{i-1})$ entails o_i
  - $p(o_i \mid \tau_j) = 0$ if $\tau_j(a_{i-1})$ is inconsistent with o_i
  - $p(o_i \mid \tau_j) = ?$ otherwise

Optimal Mode Reconfiguration

$\mu_i = argmin\ c(\mu_i')\ s.t.\ \mu_i'\ in\ \mathrm{M}_i$

MI and MR as combinatorial optimization

MI
- Variables V : transitions taken by each component
- Feasibility f : consistency of resulting state with observations
- Cost c : derived from transition probabilities

MR
- Variables V : control variables of the system
- Feasibility f : the resulting state must entail the goal
- Cost c : derived from cost of control actions

Outline

- Logistics
- Planning
- Review of last week
- Incorporating Dynamics
- Model-Based Reactive Planning

Limitations of Model-based Configuration Management

- All transitions must be directly commandable.

Requires hand coding nonlocal control procedures.

```
proc close(valve)
  unless on(driver) turn-on(driver);
  send(driver,close-valve)
```

- Presumes that commands can be interleaved arbitrarily without destructive interaction or changes in effect.

close(valve); turn-off(driver) **OK**

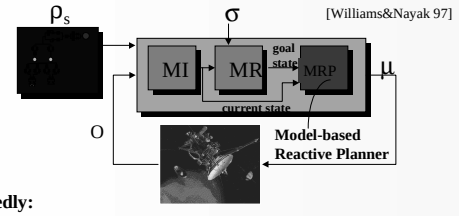
turn-off (driver); close(valve) **WRONG**

Limitations (cont.)

- MR generates a sequence of recovery actions, but there is not a tight feedback loop between MR commanding and MI monitoring.
  - MR produces task net
    E.g., close(valve);
    turn-off(driver) where
    proc close(valve)
    unless on(driver) turn-on(driver);
    send(driver,close-valve);

Model-based Reactive Execution

Model-based Reactive Executive

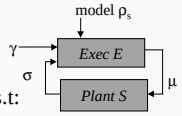


Repeatedly:

- **MI generates most likely current state**
- **MR generates least cost target state**
- **MRP generates first control action in sequence for reaching target.**
- **MI confirms desired effect of first action.**

Model-based Reactive Control System < S, Θ, E >

- S is a transition system < Π, Σ, T > with initial state Θ.
- E is a model-based reactive executive with
  - input goal configurations: $\gamma : g_0, g_1 \ldots$
  - input observables: $\sigma : o_0, o_1 \ldots$
  - output control actions $\mu : \mu_0, \mu_1 \ldots$

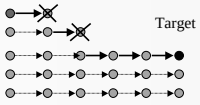


S evolves along trajectory $s : s_0, s_1 \ldots$ s.t:

- $s_0 = \Theta$
- $\sigma_i \in \Sigma$, and agrees with o_i, μ_i and s_i
- $s_{i+1} = \tau(\sigma_i)$
  - results from a failure transition, or
  - satisfies g_i or
  - **$<s_i, s_{i+1}>$ is the prefix of a simple (loop free) nominal trajectory that ends in a state satisfying g_i.**

Why plan myopically by assuming the most likely state is correct?

- Reduces computational cost.
- MI gathers information about action affects that confirm or deny this assumption.
- If the most likely state is incorrect, MI moves to a less likely state by elimination
- Eventually the correct state is reached.
- If no irreversible action is taken, this strategy will eventually reach the target, given that it was initially reachable



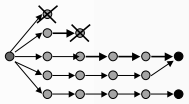
REQUIREMENT 1:

*MRP only considers reversible control actions **except** when the only effect is to repair failures.*

How does MR search efficiently over the set of feasible target states?

- How do we enumerate the set of least cost states without generating all possible plans?
- Can the reachable target states be efficiently enumerated by increasing cost?

Target achieves $\mathbf{g_i}$

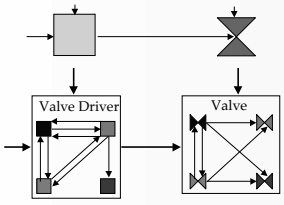


Solution falls out of the development of the reactive planner

Model-based Reactive Planning Relation to STRIPS planning?

STRIPS Operators: **PRE:** clear(hand) and on(A,B)
EFF: Delete on (A,B), clear(hand)
Add holding(A)

Model-based Reactive Planning:



Partially observability
exogenous effects
indirect control
concurrent

Comparing MRP and STRIPS

STRIPS Planning

- Action representation
  - strips operators with precondition and add/delete as effects.
- State variables only change directly by operator add/delete.
- Operators are invoked directly
- State is held constant when operators are not invoked.
- Operators are invoked one at a time.

Model-based Reactive Planning

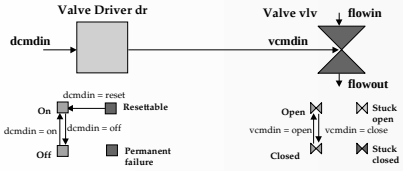
- Action representation
  - state transitions ρ_τ
  - co-temporal interactions ρ_Σ
- State variables change through transitions or through interactions.
- Transitions are controlled by establishing control values which interact with internal variables.
- State changes may not be preventable.
- Enabling one transition may necessarily cause a second transition to occur.

How Burton Achieves Reactivity

- Model compilation eliminates cotemporal interactions ρ_Σ, pre-solving NP hard part.
  - Compile transitions into a compact set of concurrent policies.
- Exploit fact that hardware typically behaves like STRIPS ops.
  - individual controllability & persistence
- Exploit requirement: planner avoid damaging effect.
- Exploit causal, loop-free structure of hardware topology.

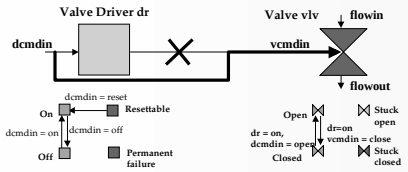
Burton: Plans first action in average O(1) time.

Driver Valve Example



- dr = resettable & dcmdin = reset ⇒ NEXT (dr1 = on)
- dr1 = on & dcmdin = open ⇒ NEXT(vcmdin = open)
- vlv = closed & vcmdin = open ⇒ NEXT(vlv = open)
- vlv = open & flowin = pos ⇒ NEXT(flowout = pos)

Model Compilation



Idea:

Eliminate hidden variables (vcmdin) and cotemporal interactions ρ_Σ,
resulting in transitions that depend only on control variables (dcmdin) and state variables (dr,vlv).

Compile by Generating Prime Implicates

- Compiled transitions are all formula of the form

  $$\Phi_i \Rightarrow \text{next}(y_i = e_i)$$

  implied by the original transition specification, where Φ_i is a smallest conjunction without hidden variables.
- E.g. vlv=closed & vcmdin=open $\Rightarrow$ NEXT (vlv=open)
  dr=on & dcmdin=open $\Rightarrow$ NEXT(vcmdin=open)

  Compiles to:

  vlv=closed & dr=on & dcmdin=open $\Rightarrow$ NEXT(vlv = open)
- 40 seconds for 12,000 clause spacecraft model.

Simplifying to Strips

- Difference 1: Transitions can occur w/o control actions.
  - tub=empty & faucet=on $\Rightarrow$ **NEXT** (tub = non-empty)
- Requirement 1:
  - Each control variable has an idling assignment.
  - No idling assignment appears in any transition.
  - The antecedent of every transition includes a non-idling control assignment.

Simplifying to Strips (cont.)

- Difference 2: Control actions can invoke multiple transitions.
  - vlv1 = closed & dr = on & dcmdin = open ⇒ NEXT (vlv1 = open)
  - vlv2 = closed & dr = on & dcmdin = open ⇒ NEXT (vlv2 = open)
- Defn: The control(state) conditions of a transition are the control(state) var assignments of antecedent.
  - state condition: vlv1 = closed & dr = on
  - control condition: dcmdin = open ⇨
- Requirement 2:
  - No set of control conditions of one transition is a proper subset of the control conditions of a different transition.

Reasons Search is Needed

1) An achieved goal can be clobbered by a subsequent goal.
   - e.g., achieving dr = off and then vlv = open clobbers dr = off.

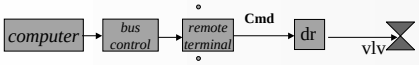
2) Two goals compete for the same variable in their subgoals.
   - e.g., latch1 and latch2 compete for the position of switch sw.

3) A state transition of a subgoal var has irreversible effect.
   - e.g., assume sw can be used once, then latch1 must be latched before latch2. ⇨

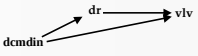
To achieve reactivity we eliminate all forms of search.

Exploiting Causality to Avoid Threats

- Observation: feedback loops rare in component schematics.



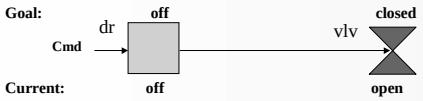
- The *Causal Graph* G is a directed graph whose vertices are state variables. G contains an edge from v1 to v2 if v1 occurs in the antecedent of v2's transition.



Requirement 3: The causal graph must be acyclic.

Exploiting Causality to Avoid Threats

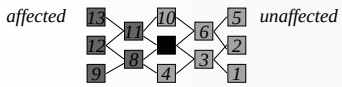
Idea: Achieve goals by working ***from effects to causes*** (e.g., vlv then dr), completing one goal before starting the next.



- work on vlv = closed
  - work on dr = on
    - next-action: Cmd = dr-on
  - next action: Cmd = vlv-close
- work on dr = off
  - next action: Cmd = dr-off

Avoiding Clobbering Sibling Goals

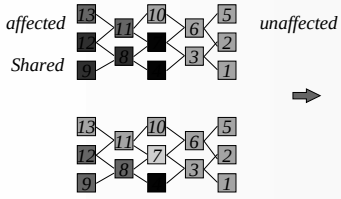
- Only vars necessary to achieve y=e are the ancestors of y; y can be changed w/o affecting descendants.



- To avoid clobbering achieved goals, Burton solves goals in an upstream order.
- Upstream order corresponds to achieving goals in order of increasing depth first number.

Avoiding Clobbering Sibling Goals

- Shared ancestors of sibling goals are required to establish both goals.



- Ancestors are no longer needed once goal has been satisfied.

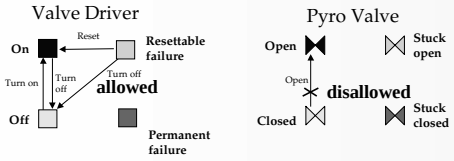
Burton: Online Algorithm *(partial)*

NextAction(initial state θ, target state γ, compiled system S')

- **Select unachieved goal:** Find unachieved goal assignment with the lowest topological number. Otherwise, return **Success.**
- **Select next transition:** Let t_y be the transition graph in S for goal variable y. *Nondeterministically* select a path p along transitions in t_y from e_i to e_f. Let SC and CC be the state and control conditions of the first transition along p.
- **Enable transition:** Control = NextAction(θ,SC,S'). If Control = **Success** then state conditions SC are already satisfied, return CC to effect transition. If **Failure** return it. Otherwise Control contains control assignments to progress on SC. Return Control.

Exploiting Safety

- Requirement 4: Only reversible transitions are allowed, except when repairing a component.

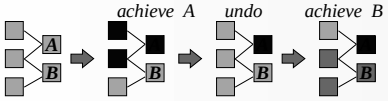


Rationale: Irreversible actions expend non-renewable resources. Require careful (human?) deliberation.

Avoiding Deadend (Sub) Goals

Lemma:

– A & B is reachable from θ by reversible transitions exactly when A and B are separately reachable from θ by reversible transitions.



Idea:

– Precompute and label all assignments that can be **reversibly** achieved from initial state θ.

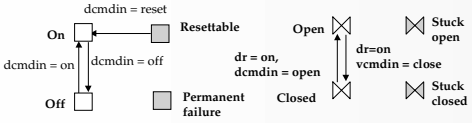
– Only use **reversible** assignments as (sub)goal, and transitions involving **reversible** assignments.

– Use Lemma to test if top-level goals are achievable.

Defining Reversibility

Definition:

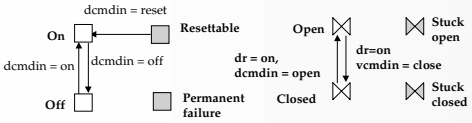
- An assignment $y = e_k$ can be **Reversibly** achieved starting at $y = e_i$ if there exists a path along **Allowed** transitions from initial value e_i to e_k and back.
- A transition is **Allowed** if all its state conditions are **Reversible**.



Reversibility Labeling Algorithm

LabelSystem(initial state θ, compiled system S')
For each state variable y of S' in decreasing topological order:

- For each transition τ_y of y, label τ_y **Allowed** if all its state conditions are labeled **Reversible**.
- Compute the strongly connected components (SCCs) of the **Allowed** transitions of y.
- Find y's initial value $y = e_i$ in θ. Label each assignment in the SCC of $y = e_i$ as **Reversible**.



Burton: Online Algorithm

NextAction(initial state θ, target state γ, system S', top?)

- **Solvable goals?:** When top?=True**,** unless each goal g in γ is labeled **Reversible**, return **Failure.**
- **Select unachieved goal:** Find unachieved goal assignment with the lowest topological number. If all achieved return **Success.**
- **Select next transition:** Let t_y be the transition graph in S for goal variable y. *Nondeterministically*, select a path p in t_y from e_i to e_f along transitions labeled **Allowed**. Let SC and CC be the state and control conditions of the first transition along p.
- **Enable transition:** Control = NextAction(θ,SC,S',False). If Control = **Success** then state conditions SC are already satisfied, return CC to effect transition. Otherwise Control contains control assignments to progress on SC. Return Control.

Incorporating Repair Actions

Definition: A repair is a transition from a failure assignment to a nominal assignment.

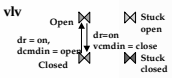
Idea:

- Burton never uses a failure assignment to achieve a goal if the failure is repairable.
- Repair minimizes irreversible effects. If y is assigned failure e_f, Burton traverses allowed transitions from e_f to the first nominal assignment reached (nominal SCC w lowest number).
- If a failure assignment is not repairable then it can be used.

Eliminating Cost of Finding Transition Paths: Generating Concurrent Policies

- NextAction is O(e*m) where
  - e is the number of transitions for a single variable y.
  - m is the maximum depth in the causal graph.

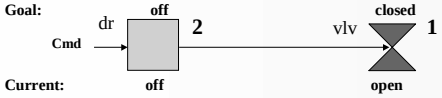
  ⇨ table lookup
- Compute a feasible policy $\pi_y(e_i,e_f)$ for variable y, where
  - e_i is a current assignment
  - e_f is a goal assignment
  - $\pi_y(e_i,e_f)$ returns the sorted conditions of the first transition along a path from e_i to e_f.



⇨

Current \ Goal	open	closed
open	Idle	dr = on dcmdin=close
closed	dr = on dcmdin=open	Idle
stuck	Failure	Failure

Burton computes next action (step 1)



Current \ Goal	on	off
on	idle	dcmdin = off
off	dcmdin = on	idle
reset failure	dcmdin = reset	dcmdin = reset

Current \ Goal	open	closed
open	idle	dr = on dcmdin=close
closed	dr = on dcmdin=open	idle
stuck	fail	fail

⇨ **dcmdin= on**

Burton computes next action (step 2)

Current \ Goal	on	off
on	idle	dcmdin = off
off	dcmdin = on	idle
reset failure	dcmdin = reset	dcmdin = reset

Current \ Goal	open	closed
open	idle	dr = on , cmdin = close
closed	dr = on, cmdin = open	idle
stuck	fail	fail

cmdin = close

Failure occurs during plan execution
Burton computes next action - step 3

Current \ Goal	on	off
on	idle	dcmdin = off
off	dcmdin = on	idle
reset failure	dcmdin = reset	dcmdin = reset

Current \ Goal	open	closed
open	idle	dr = on , cmdin = close
closed	dr = on, cmdin = open	idle
stuck	fail	fail

dcmdin = reset

Burton computes next action (step 4)

Current \ Goal	on	off
on	idle	dcmdin = off
off	dcmdin = on	idle
reset failure	dcmdin = reset	dcmdin = reset

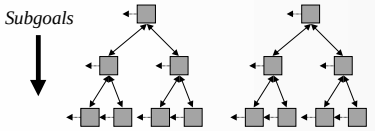
Current \ Goal	open	closed
open	idle	dr = on , cmdin = close
closed	dr = on, cmdin = open	idle
stuck	fail	fail

dcmdin = off

Complexity: Constant Average Cost

Cost of generating the first action:

- Worst Case: Maximum depth of causal graph.
- Average Cost: Constant time.
  - Each edge of the goal/subgoal tree traversed twice.
  - Each node of the goal/subgoal tree generates one action.
  - # edges < 2 * # nodes.



Outline

- Logistics
- Planning
- Review of last week
- Incorporating Dynamics
- Model-Based Reactive Planning
- Conclusion

Autonomous System Coding Challenge

Programmers must reason through system-wide interactions to generate codes for:

- monitoring
- hardware mode confirmation
- goal tracking
- detecting anomalies
- isolating faults
- diagnosing causes
- parameter estimation
- hardware reconfiguration
- fault recovery
- standby
- safing
- fault avoidance
- adaptive control
- control policy coordination

⇨ poor reuse, poor coverage, error prone

Solution: Part 1
Model-based Programming

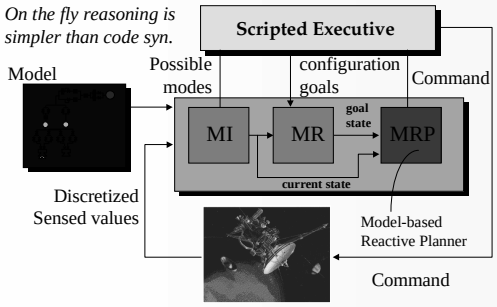
Programmers and operators generate breadth of functions from commonsense hardware models in light of mission-level goals.

Have engineers program in models, automate synthesis of code:

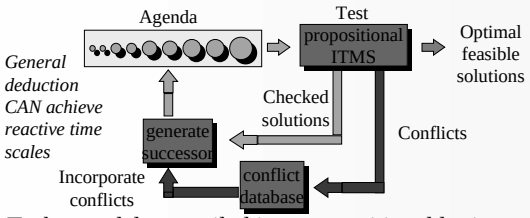
- models are compositional & highly reusable.
- generative approach covers broad set of behaviors.
- commonsense models are easy to articulate at concept stage and insensitive to design variations.

Solution: Part 2
Model-based Deductive Executive

On the fly reasoning is simpler than code syn.

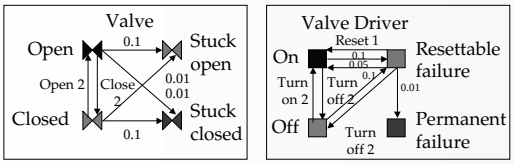


Solution: Part 3
Risc-like Best-first, Deductive Kernel



- Tasks, models compiled into propositional logic
- Conflicts dramatically focus search
- Careful enumeration grows agenda linearly
- ITMS efficiently tracks changes in truth assignments

Concurrent Transition Systems



- Represent probability of uncommanded transitions and cost of commanded transitions
- Can model
  - Reliability
  - Optimal control

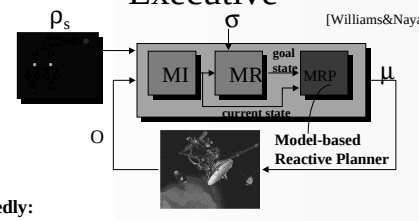
Representation with Modal Logic

- Variables $\Pi = \{mode, cmd, f_{in}, f_{out}, p_{in}, p_{out}\}$
  - $mode \in \{open, closed, stuck\text{-}open, stuck\text{-}closed\}$
- Specifying Σ with ρ_Σ
  - $mode = open \Rightarrow (p_{in} = p_{out}) \wedge (f_{in} = f_{out})$
- Specifying the nominal transition $\tau_n \in T$
  - $mode=closed \wedge cmd=open \Rightarrow NEXT\ (mode=open)$
- Specifying failure transitions
  - $mode=closed \Rightarrow NEXT\ (mode=stuck\text{-}closed)$

MI/MR as combinatorial optimization

- MI
  - variables: components with domains the possible modes
    - an assignment corresponds to a candidate diagnosis
  - feasibility: consistency with observations
  - cost: probability of a candidate diagnosis
- MR
  - variables: components with domains the possible modes
    - an assignment corresponds to a candidate repair
  - feasibility: entailment of goal
  - cost: cost of repair

Model-based Reactive Executive



Repeatedly:

- **MI generates most likely current state**
- **MR generates least cost target state**
- **MRP generates first control action in sequence for reaching target.**
- **MI confirms desired effect of first action.**

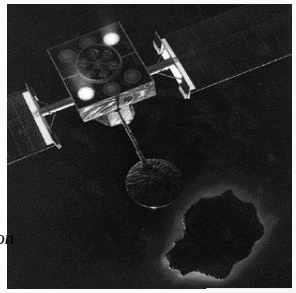
How Burton Achieves Reactivity

- Model compilation eliminates cotemporal interactions ρ_Σ, pre-solving NP hard part.
  - Compile transitions into a compact set of concurrent policies.
- Exploit fact that hardware typically behaves like STRIPS ops.
  - individual controllability & persistence
- Exploit requirement: planner avoid damaging effect.
- Exploit causal, loop-free structure of hardware topology.

Burton: Plans first action in average O(1) time.

Demonstration of Model-based Autonomy Capabilities

QuickTime™ and a Photo - JPEG decompressor are needed to see this picture



Simulated mission of Saturn orbital insertion - Fall 1995

Actual flight demonstration on Deep Space One spacecraft Fall 1998

Future: Systems that Model & Adapt

QuickTime™ and a
Photo - JPEG decompressor
are needed to see this picture

- Spontaneous learning of failure dynamics.
- Stability analysis using qualitative phase portraits.
- Large-scale nonlinear adaptive code generation from models.
- Model-based learning starting from qualitative models alone.

Future: Systems that Seek Information

- Instruments that design and execute elaborate experiments to model its environment and its own internal workings.
- Space probes that evaluate science ops and design missions.

Bioreactor:
intelligent science instrument

Future: Systems that Anticipate

QuickTime™ and a
Photo - JPEG decompressor
are needed to see this picture

- Predict critical failures for given context.
- Construct contingencies plans.
- Prepares backup resources to ensure fast response.