Hierarchical Task Networks

Planning to perform tasks rather than to achieve goals

Literature

- Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning – Theory and Practice*, chapter 11. Elsevier/Morgan Kaufmann, 2004.
- E. Sacerdoti. The nonlinear nature of plans. In: *Proc. IJCAI*, pages 206-214, 1975.
- A. Tate. Generating project networks. In: *Proc. IJCAI*, pages 888-893, 1977.

HTN Planning

- HTN planning:
  - objective: perform a given set of tasks
- input includes:
  - set of operators
  - set of methods: recipes for decomposing a complex task into more primitive subtasks
- planning process:
  - decompose non-primitive tasks recursively until primitive tasks are reached

Overview

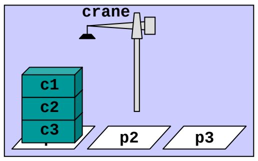
- Simple Task Networks
- HTN Planning
- Extensions
- State-Variable Representation

STN Planning

- STN: Simple Task Network
- what remains:
  - terms, literals, operators, actions, state transition function, plans
- what's new:
  - tasks to be performed
  - methods describing ways in which tasks can be performed
  - organized collections of tasks called task networks

DWR Stack Moving Example

- task: move stack of containers from pallet p1 to pallet p3 in a way the preserves the order



- (informal) methods:
  - move via intermediate: move stack to intermediate pile (reversing order) and then to final destination (reversing order again)
  - move stack: repeatedly move the topmost container until the stack is empty
  - move topmost: take followed by put action

Tasks

- task symbols: $T_S = \{t_1,\ldots,t_n\}$
  - operator names $\subsetneq T_S$: primitive tasks
  - non-primitive task symbols: T_S - operator names
- task: $t_i(r_1,\ldots,r_k)$
  - t_i: task symbol (primitive or non-primitive)
  - $r_1,\ldots,r_k$: terms, objects manipulated by the task
  - ground task: are ground
- action a accomplishes ground primitive task $t_i(r_1,\ldots,r_k)$ in state s iff
  - name(a) = t_i and
  - a is applicable in s

Simple Task Networks

- A simple task network w is an acyclic directed graph (U,E) in which
  - the node set $U = \{t_1,\ldots,t_n\}$ is a set of tasks and
  - the edges in E define a partial ordering of the tasks in U.

- A task network w is ground/primitive if all tasks $t_u \in U$ are ground/primitive, otherwise it is unground/non-primitive.

Totally Ordered STNs

- ordering: $t_u \prec t_v$ in $w=(U,E)$ iff there is a path from t_u to t_v
- STN w is totally ordered iff E defines a total order on U
  - w is a sequence of tasks: $\langle t_1,\dots,t_n\rangle$
- Let $w = \langle t_1,\dots,t_n\rangle$ be a totally ordered, ground, primitive STN. Then the plan $\pi(w)$ is defined as:
  - $\pi(w) = \langle a_1,\dots,a_n\rangle$ where $a_i = t_i$; $1 \le i \le n$

STNs: DWR Example

- tasks:
  - t_1 = take(crane,loc,c1,c2,p1): primitive, ground
  - t_2 = take(crane,loc,c2,c3,p1): primitive, ground
  - t_3 = move-stack(p1,q): non-primitive, unground
- task networks:
  - $w_1 = (\{t_1,t_2,t_3\}, \{(t_1,t_2), (t_1,t_3)\})$
    - partially ordered, non-primitive, unground
  - $w_2 = (\{t_1,t_2\}, \{(t_1,t_2)\})$
    - totally ordered: $w_2 = \langle t_1,t_2\rangle$, ground, primitive
    - $\pi(w_2) =$ ⟨take(crane,loc,c1,c2,p1),take(crane,loc,c2,c3,p1)⟩

STN Methods

- Let M_S be a set of method symbols. An STN method is a 4-tuple m=(name(m),task(m),precond(m),network(m)) where:
  - name(m):
    - the name of the method
    - syntactic expression of the form $n(x_1,\ldots,x_k)$
      - $n \in M_S$: unique method symbol
      - $x_1,\ldots,x_k$: all the variable symbols that occur in m;
  - task(m): a non-primitive task;
  - precond(m): set of literals called the method's preconditions;
  - network(m): task network (U,E) containing the set of subtasks U of m.

STN Methods: DWR Example (1)

- move topmost: take followed by put action
- take-and-put(c,k,l,p_o,p_d,x_o,x_d)
  - task: move-topmost(p_o,p_d)
  - precond: top(c,p_o), on(c,x_o), attached(p_o,l), belong(k,l), attached(p_d,l), top(x_d,p_d)
  - subtasks: $\langle$take(k,l,c,x_o,p_o),put(k,l,c,x_d,p_d)$\rangle$

STN Methods: DWR Example (2)

- move stack: repeatedly move the topmost container until the stack is empty
- recursive-move(p_o,p_d,c,x_o)
  - task: move-stack(p_o,p_d)
  - precond: top(c,p_o), on(c,x_o)
  - subtasks: ⟨move-topmost(p_o,p_d), move-stack(p_o,p_d)⟩
- no-move(p_o,p_d)
  - task: move-stack(p_o,p_d)
  - precond: top(pallet,p_o)
  - subtasks: ⟨⟩

STN Methods: DWR Example (3)

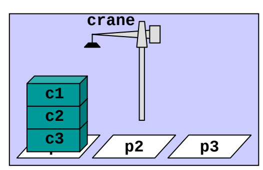
- move via intermediate: move stack to intermediate pile (reversing order) and then to final destination (reversing order again)
- move-stack-twice(p_o,p_i,p_d)
  - task: move-ordered-stack(p_o,p_d)
  - precond: -
  - subtasks: ⟨move-stack(p_o,p_i),move-stack(p_i,p_d)⟩

Applicability and Relevance

- A method instance m is <u>applicable</u> in a state s if
  - $\text{precond}^+(m) \subseteq s$ and
  - $\text{precond}^-(m) \cap s = \{\}$.
- A method instance m is <u>relevant</u> for a task t if
  - there is a substitution σ such that $\sigma(t) = \text{task}(m)$.
- The <u>decomposition</u> of a task t by a relevant method m under σ is
  - $\delta(t,m,\sigma) = \sigma(\text{network}(m))$ or
  - $\delta(t,m,\sigma) = \sigma(\langle\text{subtasks}(m)\rangle)$ if m is totally ordered.

Method Applicability and Relevance: DWR Example

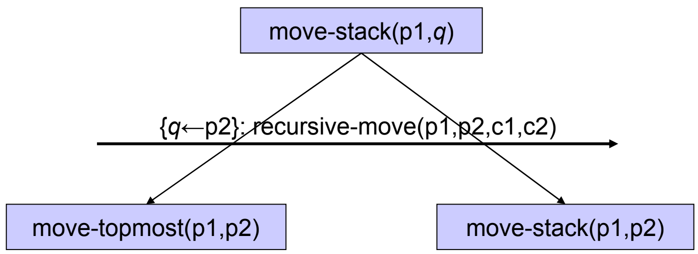
- task t = move-stack(p1,q)
- state s (as shown)



- method instance m_i = recursive-move(p1,p2,c1,c2)
  - m_i is applicable in s
  - m_i is relevant for t under $\sigma = \{q \leftarrow \text{p2}\}$

Method Decomposition: DWR Example

- $\delta(t,m_i,\sigma)$ = ⟨move-topmost(p1,p2), move-stack(p1,p2)⟩



Decomposition of Tasks in STNs

- Let
  - $w = (U,E)$ be a STN and
  - $t \in U$ be a task with no predecessors in w and
  - m a method that is relevant for t under some substitution σ with network(m) = (U_m,E_m).
- The <u>decomposition of *t* in *w* by *m* under σ</u> is the STN $\delta(w,u,m,\sigma)$ where:
  - t is replaced in U by $\sigma(U_m)$ and
  - edges in E involving t are replaced by edges to appropriate nodes in $\sigma(U_m)$.

STN Planning Domains

- An STN planning domain is a pair $\mathcal{D}=(O,M)$ where:
  - O is a set of STRIPS planning operators and
  - M is a set of STN methods.

- $\mathcal{D}$ is a total-order STN planning domain if every $m \in M$ is totally ordered.

STN Planning Problems

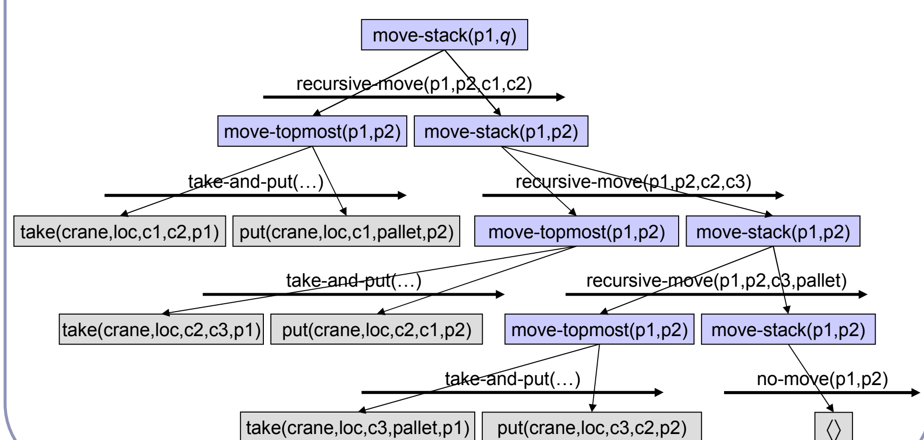
- An STN planning problem is a 4-tuple $\mathcal{P}=(s_i,w_i,O,M)$ where:
  - s_i is the initial state (a set of ground atoms)
  - w_i is a task network called the initial task network and
  - $\mathcal{D}=(O,M)$ is an STN planning domain.

- $\mathcal{P}$ is a total-order STN planning domain if w_i and $\mathcal{D}$ are both totally ordered.

STN Solutions

- A plan $\pi = \langle a_1,\ldots,a_n \rangle$ is a solution for an STN planning problem $\mathcal{P}=(s_i,w_i,O,M)$ if:
    - w_i is empty and π is empty;
- or:
    - there is a primitive task $t \in w_i$ that has no predecessors in w_i and
    - $a_1=t$ is applicable in s_i and
    - $\pi' = \langle a_2,\ldots,a_n \rangle$ is a solution for $\mathcal{P}'=(\gamma(s_i,a_1), w_i-\{t\}, O, M)$
- or:
    - there is a non-primitive task $t \in w_i$ that has no predecessors in w_i and
    - $m \in M$ is relevant for t, i.e. $\sigma(t)$ = task(m) and applicable in s_i and
    - π is a solution for $\mathcal{P}'=(s_i, \delta(w_i,t,m,\sigma), O, M)$.

Decomposition Tree: DWR Example



Ground-TFD: Pseudo Code

function Ground-TFD(s,$\langle t_1,\ldots,t_k\rangle$,$O$,$M$)
  if k=0 **return** $\langle\rangle$
  if t_1.isPrimitive() **then**
    actions = $\{(a,\sigma) \mid a=\sigma(t_1)$ and a applicable in $s\}$
    if *actions*.isEmpty() **then return** failure
    (a,σ) = *actions*.chooseOne()
    plan ← Ground-TFD($\gamma(s,a)$,$\sigma(\langle t_2,\ldots,t_k\rangle)$,$O$,$M$)
    if *plan* = failure **then return** failure
    else return $\langle a\rangle$ • *plan*
  else
    methods = $\{(m,\sigma) \mid m$ is relevant for $\sigma(t_1)$ and m is applicable in $s\}$
    if *methods*.isEmpty() **then return** failure
    (m,σ) = *methods*.chooseOne()
    plan ← subtasks(m) • $\sigma(\langle t_2,\ldots,t_k\rangle)$
    return Ground-TFD(s,*plan*,O,M)

TFD vs. Forward/Backward Search

- choosing actions:
  - TFD considers only applicable actions like forward search
  - TFD considers only relevant actions like backward search
- plan generation:
  - TFD generates actions execution order; current world state always known
- lifting:
  - Ground-TFD can be generalized to Lifted-TFD resulting in same advantages as lifted backward search

Ground-PFD: Pseudo Code

function Ground-PFD(s,w,O,M)
    if $w.U$={} **return** ⟨⟩
    $task$ ← {$t \in U$ | t has no predecessors in $w.E$}.chooseOne()
    if $task$.isPrimitive() **then**
        $actions$ = {(a,σ) | a=σ(t_1) and a applicable in s}
        if $actions$.isEmpty() **then return** failure
        (a,σ) = $actions$.chooseOne()
        $plan$ ← Ground-PFD(γ(s,a),σ(w-{$task$}),O,M)
        if $plan$ = failure **then return** failure
        else return ⟨a⟩ • $plan$
    else
        $methods$ = {(m,σ) | m is relevant for σ(t_1) and m is applicable in s}
        if $methods$.isEmpty() **then return** failure
        (m,σ) = $methods$.chooseOne()
        return Ground-PFD(s, δ(w,$task$,m,σ),O,M)

Overview

- Simple Task Networks
- HTN Planning
- Extensions
- State-Variable Representation

Preconditions in STN Planning

- STN planning constraints:
  - ordering constraints: maintained in network
  - preconditions:
    - enforced by planning procedure
    - must know state to test for applicability
    - must perform forward search
- HTN Planning
  - additional bookkeeping maintains general constraints explicitly

First and Last Network Nodes

- Let
  - $\pi = \langle a_1,\ldots,a_n \rangle$ be a solution for *w*,
  - $U' \subseteq U$ be a set of tasks in w, and
  - $A(U')$ the subset of actions in π such that each $a_i \in A(U')$ is a descendant of some $t \in U'$ in the decomposition tree.
- Then we define:
  - first(U',π) = the action $a_i \in A(U')$ that occurs first in π; and
  - last(U',π) = the action $a_i \in A(U')$ that occurs last in π.

Hierarchical Task Networks

- A <u>(hierarchical) task network</u> is a pair w=(U,C), where:
  - U is a set of tasks and
  - C is a set of constraints of the following types:
    - $t_1 \prec t_2$: precedence constraint between tasks satisfied if in every solution π: last({t},π) ≺ first({t},π);
    - before(U',l): satisfied if in every solution π: literal l holds in the state just before first(U',π);
    - after(U',l): satisfied if in every solution π: literal l holds in the state just after last(U',π);
    - between(U',U'',l): satisfied if in every solution π: literal l holds in every state after last(U',π) and before first(U'',π).

HTN Methods

- Let M_S be a set of method symbols. An <u>HTN method</u> is a 4-tuple m=(name(m),task(m),subtasks(m),constr(m)) where:
  - name(m):
    - the name of the method
    - syntactic expression of the form $n(x_1,\ldots,x_k)$
      - $n \in M_S$: unique method symbol
      - $x_1,\ldots,x_k$: all the variable symbols that occur in m;
  - task(m): a non-primitive task;
  - (subtasks(m),constr(m)): a task network.

HTN Methods: DWR Example (1)

- move topmost: take followed by put action
- take-and-put(c,k,l,p_o,p_d,x_o,x_d)
  - task: move-topmost(p_o,p_d)
  - network:
    - subtasks: {t_1=take(k,l,c,x_o,p_o), t_2=put(k,l,c,x_d,p_d)}
    - constraints: {$t_1 \prec t_2$, before({t_1}, top(c,p_o)), before({t_1}, on(c,x_o)), before({t_1}, attached(p_o,l)), before({t_1}, belong(k,l)), before({t_2}, attached(p_d,l)), before({t_2}, top(x_d,p_d))}

HTN Methods: DWR Example (2)

- move stack: repeatedly move the topmost container until the stack is empty
- recursive-move(p_o,p_d,c,x_o)
  - task: move-stack(p_o,p_d)
  - network:
    - subtasks: {t_1=move-topmost(p_o,p_d), t_2=move-stack(p_o,p_d)}
    - constraints: {$t_1 \prec t_2$, before({t_1}, top(c,p_o)), before({t_1}, on(c,x_o))}
- move-one(p_o,p_d,c)
  - task: move-stack(p_o,p_d)
  - network:
    - subtasks: {t_1=move-topmost(p_o,p_d)}
    - constraints: {before({t_1}, top(c,p_o)), before({t_1}, on(c,pallet))}

HTN Decomposition

- Let $w=(U,C)$ be a task network, $t \in U$ a task, and m a method such that $\sigma(\text{task}(m))=t$. Then the decomposition of t in w using m under σ is defined as:

  $\delta(w,t,m,\sigma) = ((U-\{t\}) \cup \sigma(\text{subtasks}(m)),\ C' \cup \sigma(\text{constr}(m)))$

  where C' is modified from C as follows:
  - for every precedence constraint in C that contains t, replace it with precedence constraints containing $\sigma(\text{subtasks}(m))$ instead of t; and
  - for every before-, after-, or between constraint over tasks U' containing t, replace U' with $(U'-\{t\}) \cup \sigma(\text{subtasks}(m))$.

HTN Decomposition: Example

- network: $w = (\{t_1\text{= move-stack(p1},q)\}, \{\})$
- $\delta(w, t_1, \text{recursive-move}(p_o,p_d,c,x_o), \{p_o \leftarrow \text{p1}, p_d \leftarrow q\}) = w' =$
  - $(\{t_2\text{=move-topmost(p1},q),\ t_3\text{=move-stack(p1},q)\},$
  - $\{t_2 \prec t_3,\ \text{before}(\{t_2\}, \text{top}(c,\text{p1})),\ \text{before}(\{t_2\}, \text{on}(c,x_o))\})$
- $\delta(w', t_2, \text{take-and-put}(c,k,l,p_o,p_d,x_o,x_d), \{p_o \leftarrow \text{p1}, p_d \leftarrow q\}) =$
  - $(\{t_3\text{=move-stack(p1},q),\ t_4\text{=take}(k,l,c,x_o,\text{p1}),\ t_5\text{=put}(k,l,c,x_d,q)\},$
  - $\{t_4 \prec t_3,\ t_5 \prec t_3,\ \text{before}(\{t_4,t_5\}, \text{top}(c,\text{p1})),\ \text{before}(\{t_4,t_5\}, \text{on}(c,x_o))\} \cup \{t_4 \prec t_5,\ \text{before}(\{t_4\}, \text{top}(c,\text{p1})),\ \text{before}(\{t_4\}, \text{on}(c,x_o)),\ \text{before}(\{t_4\}, \text{attached}(\text{p1},l)),\ \text{before}(\{t_4\}, \text{belong}(k,l)),\ \text{before}(\{t_5\}, \text{attached}(q,l)),\ \text{before}(\{t_5\}, \text{top}(x_d,q))\})$

HTN Planning Domains and Problems

- An <u>HTN planning domain</u> is a pair $\mathcal{D}=(O,M)$ where:
  - O is a set of STRIPS planning operators and
  - M is a set of HTN methods.
- An <u>HTN planning problem</u> is a 4-tuple $\mathcal{P}=(s_i,w_i,O,M)$ where:
  - s_i is the initial state (a set of ground atoms)
  - w_i is a task network called the <u>initial task network</u> and
  - $\mathcal{D}=(O,M)$ is an HTN planning domain.

Solutions for Primitive HTNs

- Let (U,C) be a primitive HTN. A plan $\pi = \langle a_1,\ldots,a_n\rangle$ is a solution for $\mathcal{P}=(s_i,(U,C),O,M)$ if there is a ground instance $(\sigma(U),\sigma(C))$ of (U,C) and a total ordering $\langle t_1,\ldots,t_n\rangle$ of tasks in $\sigma(U)$ such that:
  - for i=1…n: name(a_i) = t_i;
  - π is executable in s_i, i.e. $\gamma(s_i,\pi)$ is defined;
  - the ordering of $\langle t_1,\ldots,t_n\rangle$ respects the ordering constraints in $\sigma(C)$;
  - for every constraint before(U',l) in $\sigma(C)$ where t_k=first(U',π): l must hold in $\gamma(s_i, \langle a_1,\ldots,a_{k-1}\rangle)$;
  - for every constraint after(U',l) in $\sigma(C)$ where t_k=last(U',π): l must hold in $\gamma(s_i, \langle a_1,\ldots,a_k\rangle)$;
  - for every constraint between(U',U'',l) in $\sigma(C)$ where t_k=first(U',π) and t_m=last(U'',π): l must hold in every state $\gamma(s_i, \langle a_1,\ldots,a_j\rangle)$, $j\in\{k\ldots m-1\}$.

Solutions for Non-Primitive HTNs

- Let $w = (U,C)$ be a non-primitive HTN. A plan $\pi = \langle a_1,\ldots,a_n \rangle$ is a solution for $\mathcal{P}=(s_i,w,O,M)$ if there is a sequence of task decompositions that can be applied to w such that:
  - the result of the decompositions is a primitive HTN w'; and
  - π is a solution for $\mathcal{P}'=(s_i,w',O,M)$.

Abstract-HTN: Pseudo Code

function Abstract-HTN(s,U,C,O,M)
  if (U,C).isInconsistent() **then return** failure
  if U.isPrimitive() **then**
    return extractSolution(s,U,C,O)
  else
    return decomposeTask(s,U,C,O,M)

extractSolution: Pseudo Code

function extractSolution(s,U,C,O)
$\langle t_1,\ldots,t_n \rangle \leftarrow U$.chooseSequence($C$)
$\langle a_1,\ldots,a_n \rangle \leftarrow$
$\langle t_1,\ldots,t_n \rangle$.chooseGrounding($s$,$C$,$O$)
if $\langle a_1,\ldots,a_n \rangle$.satisfies($C$) **then**
return $\langle a_1,\ldots,a_n \rangle$
return failure

decomposeTask: Pseudo Code

function decomposeTask(s,U,C,O,M)
$t \leftarrow U$.nonPrimitives().selectOne()
methods $\leftarrow \{(m,\sigma) \mid m \in M$ and $\sigma(\text{task}(m)) = \sigma(t)\}$
if *methods*.isEmpty() **then return** failure
$(m,\sigma) \leftarrow$ *methods*.chooseOne()
$(U',C') \leftarrow \delta((U,C),t,m,\sigma)$
$(U',C') \leftarrow (U',C')$.applyCritic()
return Abstract-HTN(s,U',C',O,M)

HTN vs. STRIPS Planning

- Since
  - HTN is generalization of STN Planning, and
  - STN problems can encode undecidable problems, but
  - STRIPS cannot encode such problems:
- **STN/HTN formalism is more expressive**
- non-recursive STN can be translated into equivalent STRIPS problem
  - but exponentially larger in worst case
- “regular” STN is equivalent to STRIPS

Overview

- Simple Task Networks
- HTN Planning
- ➡ Extensions
- State-Variable Representation

Functions in Terms

- allow function terms in world state and method constraints
- ground versions of all planning algorithms may fail
  - potentially infinite number of ground instances of a given term
- lifted algorithms can be applied with most general unifier
  - least commitment approach instantiates only as far as necessary
  - plan-existence may not be decidable

Axiomatic Inference

- use theorem prover to infer derived knowledge within world states
  - undecidability of first-order logic in general
- idea: use restricted (decidable) subset of first-order logic: Horn clauses
  - only positive preconditions can be derived
  - precondition *p* is satisfied in state *s* iff *p* can be proved in *s*

Attached Procedures

- associate predicates with procedures
- modify planning algorithm
  - evaluate preconditions by
    - calling the procedure attached to the predicate symbol if there is such a procedure
    - test against world state (set-relation, theorem prover) otherwise
- soundness and completeness: depends on procedures

High-Level Effects

- allow user to declare effects for non-primitive methods
- aim:
  - establish preconditions
  - prune partial plans if high-level effects threaten preconditions
- increases efficiency
- problem: semantics

Other Extensions

- other constraints
  - time constraints
  - resource constraints
- extended goals
  - states to be avoided
  - required intermediate states
  - limited plan length
  - visit states multiple times

Overview

- Simple Task Networks
- HTN Planning
- Extensions
- State-Variable Representation

State Variables

- some relations are functions
  - example: at(r1,loc1): relates robot r1 to location loc1 in some state
    - truth value changes from state to state
    - will only be true for exactly one location *l* in each state
- idea: represent such relations using state-variable functions mapping states into objects
  - example: functional representation: rloc:robots×S→locations

States in the State-Variable Representation

- Let X be a set of state-variable functions. A k-ary state variable is an expression of the form $x(v_1,\ldots v_k)$ where:
  - $x \in X$ is a state-variable function and
  - v_i is either an object constant or an object variable.
- A state-variable state description is a set of expressions of the form $x_s=c$ where:
  - x_s is a ground state variable $x(v_1,\ldots v_k)$ and
  - c is an object constant.

DWR Example: State-Variable State Descriptions

- simplified: no cranes, no piles
- state-variable functions:
  - rloc: robots×S → locations
  - rolad: robots×S→containers ∪ {nil}
  - cpos: containers×S → locations ∪ robots
- sample state-variable state descriptions:
  - {rloc(r1)=loc1, rload(r1)=nil, cpos(c1)=loc1, cpos(c2)=loc2, cpos(c3)=loc2}
  - {rloc(r1)=loc1, rload(r1)=c1, cpos(c1)=r1, cpos(c2)=loc2, cpos(c3)=loc2}

Operators in the State-Variable Representation

- A state-variable planning operator is a triple (name(o), precond(o), effects(o)) where:
  - name(o) is a syntactic expression of the form $n(x_1,\ldots,x_k)$ where n is a (unique) symbol and $x_1,\ldots,x_k$ are all the object variables that appear in o,
  - precond(o) are the unions of a state-variable state description and some rigid relations, and
  - effects(o) are sets of expressions of the form $x_s \leftarrow v_{k+1}$ where:
    - x_s is a ground state variable $x(v_1,\ldots v_k)$ and
    - v_{k+1} is an object constant or an object variable.

DWR Example: State-Variable Operators

- move(r,l,m)
  - precond: rloc(r)=l, adjacent(l,m)
  - effects: rloc(r)←m
- load(r,c,l)
  - precond: rloc(r)=l, cpos(c)=l, rload(r)=nil
  - effects: cpos(c)←r, rload(r)←c
- unload(r,c,l)
  - precond: rloc(r)=l, rload(r)=c
  - effects: rload(r)←nil, cpos(c)←l

Applicability and State Transitions

- Let a be an action and s a state. Then a is applicable in s iff:
  - all rigid relations mentioned in precond(a) hold, and
  - if $x_s=c \in$ precond(a) then $x_s=c \in s$.
- The state transition function γ for an action a in state s is defined as $\gamma(s,a) = \{x_s=c \mid x\in X\}$ where:
  - $x_s \leftarrow c \in$ effects(a) or
  - $x_s=c \in s$ otherwise.

State-Variable Planning Domains

- Let X be a set of state-variable functions. A state-variable planning domain on X is a restricted state-transition system $\Sigma=(S,A,\gamma)$ such that:
  - S is a set of state-variable state descriptions,
  - A is a set of ground instances of some state-variable planning operators O,
  - $\gamma: S\times A\rightarrow S$ where
    - $\gamma(s,a)=\{x_s=c \mid x\in X$ and $x_s\leftarrow c \in$ effects(a) or $x_s=c \in s$ otherwise$\}$ if a is applicable in s
    - $\gamma(s,a)$=undefined otherwise,
  - S is closed under γ

State-Variable Planning Problems

- A state-variable planning problem is a triple $\mathcal{P}=(\Sigma,s_i,g)$ where:
  - $\Sigma=(S,A,\gamma)$ is a state-variable planning domain on some set of state-variable functions X
  - $s_i\in S$ is the initial state
  - g is a set of expressions of the form $x_s=c$ describing the goal such that the set of goal states is: $S_g=\{s\in S \mid x_s=c \in s\}$

Relevance and Regression Sets

- Let $\mathcal{P}=(\Sigma,s_i,g)$ be a state-variable planning problem. An action $a \in A$ is relevant for g if
  - $g \cap \text{effects}(a) \neq \{\}$ and
  - for every $x_s=c \in g$, there is no $x_s \leftarrow d \in \text{effects}(a)$ such that $c \neq d$.
- The regression set of g for a relevant action $a \in A$ is:
  - $\gamma^{-1}(g,a)=(g - \vartheta(a)) \cup \text{precond}(a)$ where
  - $\vartheta(a) = \{x_s=c \mid x_s \leftarrow c \in \text{effects}(a)\}$
- definition for all regression sets $\Gamma^{<}(g)$ exactly as for propositional case

Statement of a State-Variable Planning Problem

- A statement of a state-variable planning problem is a triple $P=(O,s_i,g)$ where:
  - O is a set of planning operators in an appropriate state-variable planning domain $\Sigma=(S,A,\gamma)$ on X
  - s_i is the initial state in an appropriate state-variable planning problem $\mathcal{P}=(\Sigma,s_i,g)$
  - g is a goal in the same state-variable planning problem $\mathcal{P}$

Translation: STRIPS to State-Variable Representation

- Let $P=(O,s_i,g)$ be a statement of a classical planning problem. In the operators O, in the initial state s_i, and in the goal g:
  - replace every positive literal $p(t_1,\ldots,t_n)$ with a state-variable expression $p(t_1,\ldots,t_n)=1$ or $p(t_1,\ldots,t_n)\leftarrow 1$ in the operators' effects, and
  - replace every negative literal $\neg p(t_1,\ldots,t_n)$ with a state-variable expression $p(t_1,\ldots,t_n)=0$ or $p(t_1,\ldots,t_n)\leftarrow 0$ in the operators' effects.

Translation: State-Variable to STRIPS Representation

- Let $P=(O,s_i,g)$ be a statement of a state-variable planning problem. In the operators' preconditions, in the initial state s_i, and in the goal g:
  - replace every state-variable expression $p(t_1,\ldots,t_n)=v$ with an atom $p(t_1,\ldots,t_n,v)$, and
- in the operators' effects:
  - replace every state-variable assignment $p(t_1,\ldots,t_n)\leftarrow v$ with a pair of literals $p(t_1,\ldots,t_n,v)$, $\neg p(t_1,\ldots,t_n,w)$, and add $p(t_1,\ldots,t_n,w)$ to the respective operators preconditions.

Overview

- Simple Task Networks
- HTN Planning
- Extensions
- State-Variable Representation