Translating HTNs to PDDL

A Small Amount of Domain Knowledge Can Go a Long Way

Ron Alford Ugur Kuter Dana Nau

University of Maryland, College Park

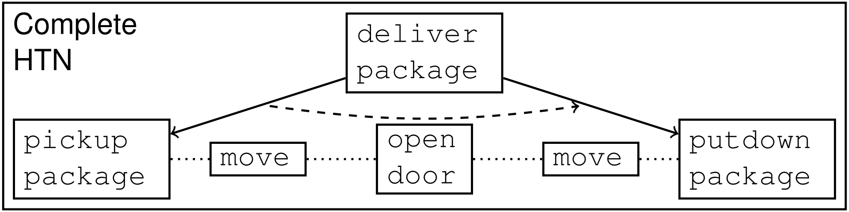
IJCAI-09 Technical Presentation

Motivation

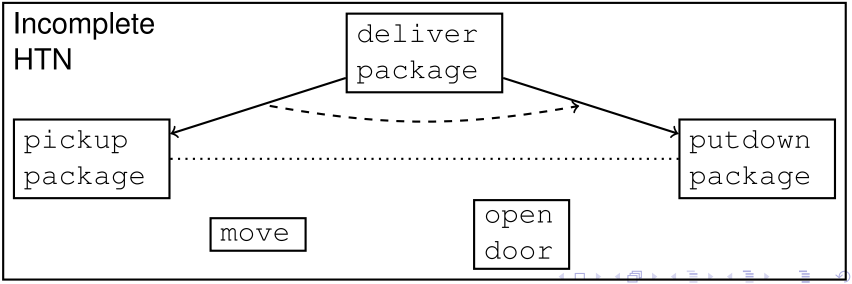
- HTNs allow domain authors to incorporate domain specific information on how to find solutions
- Traditional HTN planners need everything fully specified.
  - They require complete HTN domain descriptions
  - Otherwise they cannot generate solutions to the domain's planning problems.
- Writing HTN descriptions can be a complicated task

```
 (weight-v ?vehicle ?weight-v)
 (or (and (not (expected ?vehicle ?route ?value))
    (call <= (call + ?weight-v ?weight) ?weight-capr))
     (and (expected ?vehicle ?route ?value)
    (call <= (call + (call + ?weight-v ?weight) ?value) ?weight-capr)))
 (call >= ?height-capr ?height))
(:ordered (set-next ?vehicle ?origin)
    (:immediate add-exp-weight ?vehicle ?route ?weight)
    (:immediate at-vehicle ?vehicle ?origin)
    (:immediate !!delete-protection (next ?vehicle ?origin))
    (:immediate set-next ?vehicle ?destination)
    (:immediate load ?package ?vehicle ?origin)
    (move-vehicle-non-road ?vehicle ?origin ?destination ?route)
    (:immediate !!delete-protection (next ?vehicle ?destination))
    (:immediate del-exp-weight ?vehicle ?route ?weight)
    (:immediate unload ?package ?vehicle ?destination)))
```

Simple Example: An Office Delivery Task



- Allow domain author to specify only part of the HTN
- Would rather let the planner figure out how to insert `move` and `open door` actions



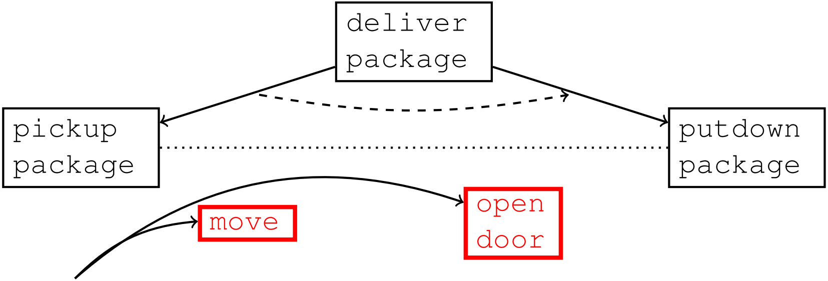
Contributions

- An automatic translation from HTNs to PDDL
  - Classical solutions correspond to valid HTN decompositions
  - Translation runs in linear time & space
- Can translate *incomplete* HTN domain descriptions:
  - Domain descriptions that don't contain enough for an HTN planner to work on
  - Classical planner can fill in the details in its own way
- Experiments using the Fast Forward (FF) planner.
  - Even small amounts of HTN knowledge substantially improved FF's running time

Outline

- Translate each HTN operator
- Translate each HTN method
- Experimental Results

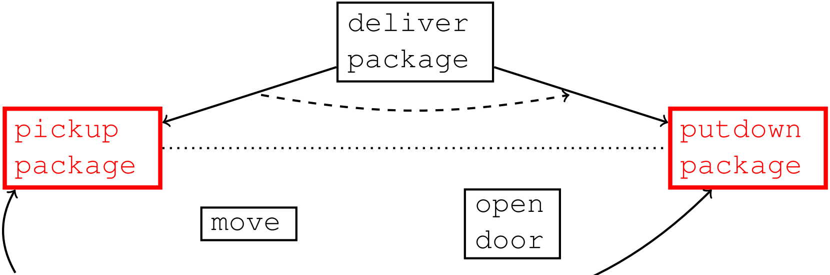
Translating Uncontrolled Operators



- Some operators are uncontrolled by the HTNs
  - I.e., they are missing from the HTN domain description
  - Classical planner can insert these at any point during the decomposition

- These operators pass through the translator unchanged

Translating Controlled Operators

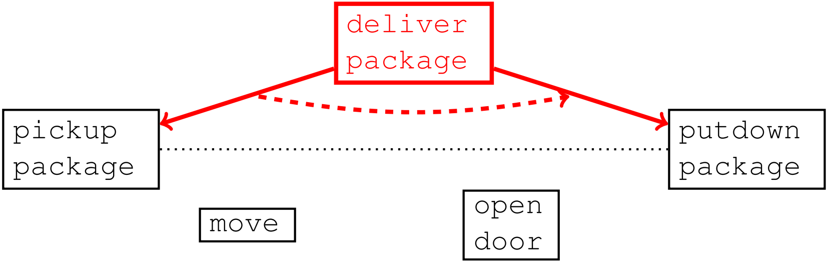


- For controlled operators
  - Can only insert operator when called as a subtask of a method
  - Need to prevent planner from inserting operator until it's called
- How to achieve this? A *control predicate*
  - Added as a precondition to operator
  - Asserted by a calling methods
  - Retracted when operator fires

Operator: $pickup(p, i)$

$$\begin{aligned}\text{name} &= pickup(p,i)\\ \text{pre} &= \{\text{do}_{pickup}(p,i),\\ &\qquad \ldots\}\\ \text{eff} &= \{\neg\text{do}_{pickup}(p,i),\\ &\qquad \ldots\}\end{aligned}$$

Translating Methods: Overview



- When translating a method, we need to:
    - Check the method's preconditions
    - Manage the subtasks as they decompose
- Split the control structure over multiple operators

Method: *deliver*(p, i, g)

$$\begin{aligned} \text{name} &= \textit{deliver}(p, i, g) \\ \text{pre} &= \{\ldots\} \\ \text{subtasks} &= \{\textit{pickup}(p, i), \\ &\qquad \textit{putdown}(p, g)\} \end{aligned}$$

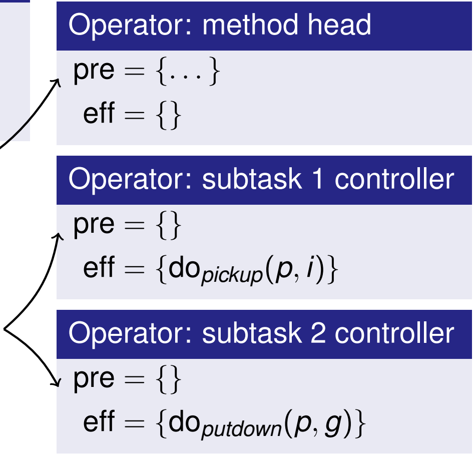
Translating Methods: Control Scaffolding

Method: $deliver(p, i, g)$

$$\text{pre} = \{\dots\}$$

$$\text{subtasks} = \{pickup(p, i),\; putdown(p, g)\}$$

- **One operator for the method head**
  - Checks method preconditions
- **One controller operator for each subtask**
  - Each controller starts a subtask
  - Uses *control predicates* mentioned earlier



Operator: method head

$\text{pre} = \{\dots\}$

$\text{eff} = \{\}$

Operator: subtask 1 controller

$\text{pre} = \{\}$

$\text{eff} = \{\text{do}_{pickup}(p, i)\}$

Operator: subtask 2 controller

$\text{pre} = \{\}$

$\text{eff} = \{\text{do}_{putdown}(p, g)\}$

Translating Methods: Control Relationships

Method: $deliver(p, i, g)$

$$\text{pre} = \{\ldots\}$$
$$\text{subtasks} = \{pickup(p, i),\; putdown(p, g)\}$$

- Method head operator instantiates parameters for subtasks
- Each subtask controller starts next task

Operator: method head

$$\text{pre} = \{\text{do}_{deliver}(p, i, g), \ldots\}$$
$$\text{eff} = \{\text{do}_{subtask1}(p, i, g)\}$$

Operator: subtask 1 controller

$$\text{pre} = \{\text{do}_{subtask1}(p, i, g)\}$$
$$\text{eff} = \{\text{do}_{subtask2}(p, i, g), \ldots\}$$

Operator: subtask 2 controller

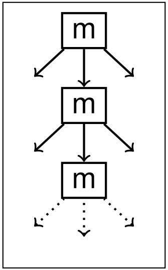
$$\text{pre} = \{\text{do}_{subtask2}(p, i, g)\}$$
$$\text{eff} = \{\ldots\}$$

Translating Methods Interlude: Recursion

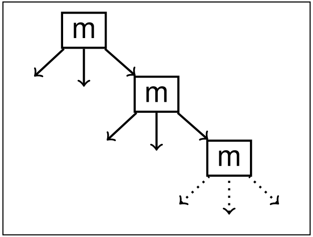
Need to distinguish between method invocations at different depths of recursion. So:

- Add a new parameter to each method: *level*
- Parameter instantiated with current recursion depth

General Recursion



Tail Recursion



Translating Methods: Dealing with Recursion

Method: *deliver*(p, i, g)

$$\begin{aligned} \text{pre} &= \{\ldots\} \\ \text{subtasks} &= \{\mathit{pickup}(p, i), \\ &\qquad \mathit{putdown}(p, g)\} \end{aligned}$$

- Add a *level* parameter to the subtask controllers
  - Instantiate to the current recursion depth
- Subtask controllers increment level before calling task
- Operators decrement level
- Last subtask operator leaves level alone
  - makes tail recursion cheap

Operator: method head

$$\begin{aligned} \text{pre} &= \{\text{do}_{\mathit{deliver}}(p, i, g), \\ &\qquad \textbf{level(v)}, \ldots\} \\ \text{eff} &= \{\text{do}_{\mathit{subtask}1}(p, i, g, \mathbf{v})\} \end{aligned}$$

Operator: subtask 1 controller

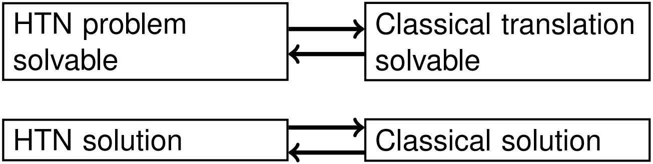
$$\begin{aligned} \text{pre} &= \{\text{do}_{\mathit{subtask}1}(p, i, g, \mathbf{v}), \\ &\qquad \textbf{level(v)}\} \\ \text{eff} &= \{\text{do}_{\mathit{subtask}2}(p, i, g, \mathbf{v}), \\ &\qquad \text{increment v}, \ldots\} \end{aligned}$$

Operator: subtask 2 controller

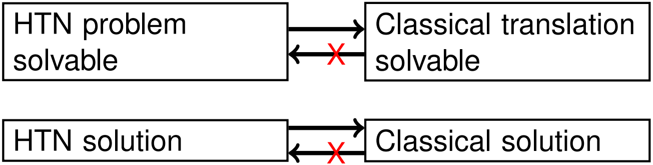
$$\begin{aligned} \text{pre} &= \{\text{do}_{\mathit{subtask}2}(p, i, g, \mathbf{v}), \\ &\qquad \textbf{level(v)}\} \\ \text{eff} &= \{\ldots\} \end{aligned}$$

Formal Properties

For complete HTN descriptions:



For incomplete HTN descriptions:



Since the HTN description is incomplete, the classical planner can come up with solutions not mentioned in the HTN

Experiments

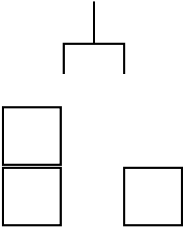
- In domains that are hard for a classical planner, how much can we improve performance via partial HTN translations?

- Took FastForward (FF) [Hoffmann and Nebel, 2001]
  - Investigated how well it did on its own
  - Investigated how it did with a translated HTN
    - Used the simplest HTNs we could find
    - Some of the HTNs were incomplete

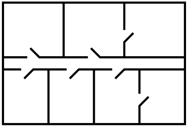
- 3 domains, about 5000 experiments:
  - Towers of Hanoi
  - Blocks World
  - An office delivery domain

Towers of Hanoi

Blocks World



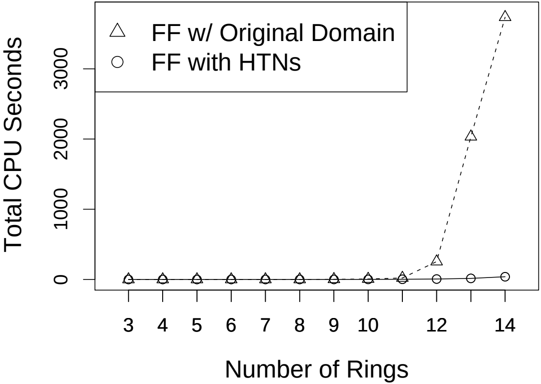
Office Delivery



Results: Towers of Hanoi

HTN Description:

- Move smallest ring to next tower
- Move whichever other ring can move
- Repeat

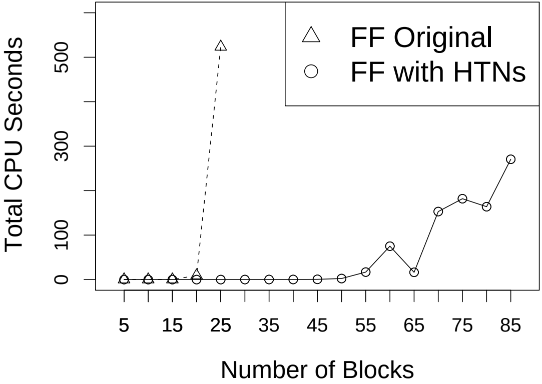


- Varying number of rings
- Each point average of 100 runs
    - Run times vary!
- FF with the translated HTNs runs exponentially better

Results: Blocks World

HTN Description

- Pick up block
- Put down block either:
  - On table
  - Final location
- Repeat

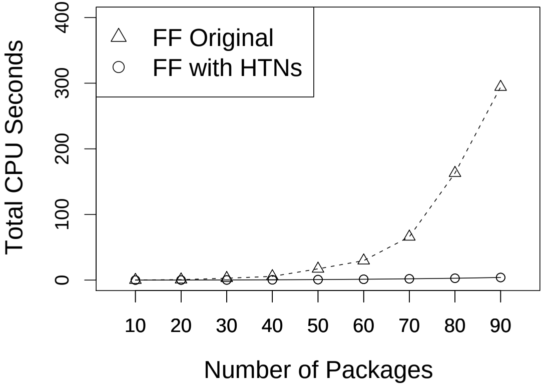


- “Complete” but inefficient HTN domain description
- Vary number of blocks, 100 problems of each size.
  - FF with translated HTNs can solve problems up to 85 blocks in size
  - Can only do 25 block problems without HTNs

Results: An Office Delivery Domain

HTN Description:

- Pick up a package
- Put down package in goal location
- repeat



- Incomplete HTN description
- HTNs order ‘pickup’/‘putdown’ actions
- ‘move’ and ‘open’ actions uncontrolled by HTNs
- Planner is free to intersperse ‘move’ and ‘open’ actions in the HTN decomposition

Conclusions

- Translation from HTNs into PDDL
- Allows domain authors to specify small amounts of HTN domain knowledge
- Translated HTNs can increase FF's performance by several orders of magnitude
    - Even when the HTNs are incomplete

Future Work

- Test against dedicated HTN planners
  - Initial results show FF outperforming SHOP2 with the Towers of Hanoi HTN domain description.
- See how well classical heuristics benefit HTN planning
- Create a native incomplete-HTN planner

PDDL Planning

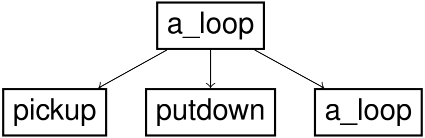
- PDDL Planning Problem: $P = (s_0, g, O)$
  - s_0 is the *initial* state
  - g is the *goal* (a set of ground literals of L)
  - O is a set of operators
- Each operator $o \in O$ is a triple: $o = (\text{name}(o), \text{pre}(o), \text{eff}(o))$
  - name(o) is o's name and argument list
  - pre(o) a formula called o's *preconditions*
  - eff(o) a set of literals called o's *effects*

PDDL Example

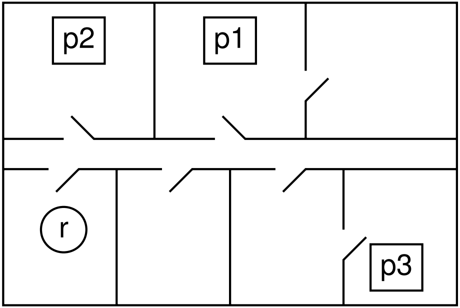
HTN Planning

HTN Planning:

- Tasks represent activities: $t(x_1, \ldots, x_q)$
  - *primitive* tasks correspond to operator names
  - *nonprimitive* tasks are implemented by methods
- Methods: $m = (\text{name}(m), \text{task}(m), \text{pre}(m), \text{subtasks}(m))$
  - Methods take a task, and decompose it into multiple subtasks
  - For our purposes, the subtasks must be completed in order



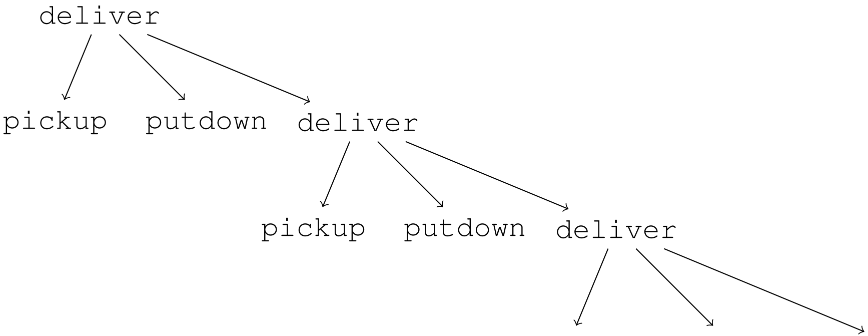
Example: Office Delivery



Actions:

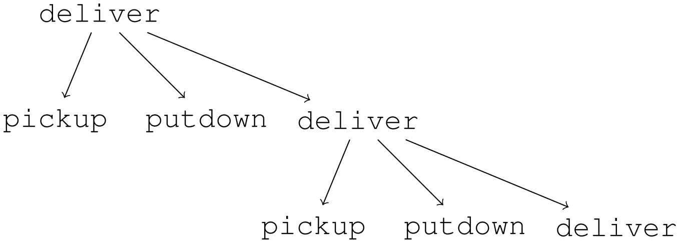
- `pickup(p, i)` - Pickup package *p* at location *i*
- `putdown(p,i)` - Putdown package *p* at location *i*
- `open(d)` - Open a door between rooms
- `move(i1, i2)` - Move between rooms

HTN Translation Restrictions



- Subtasks totally ordered
- Upper bound on depth of recursion
    - Except: tail recursion unbounded
    - Call this *non-tail height*

Non-Tail Height



- *non-tail height*: Level of method decomposition in a solution, ignoring tail decomposition.
- Often depends only on the methods, not the individual problem instances.
- Denote level by:
  - Constants $\mathrm{d}_0, \mathrm{d}_1, \ldots, \mathrm{d}_H$
  - Ordering over constants: next($\mathrm{d}_1, \mathrm{d}_2$), next($\mathrm{d}_2, \mathrm{d}_3$), . . .
  - Predicate level(d_i) for current level

Method Translation

$$\begin{aligned} \text{name} &= \mathit{deliver\_one}_{\mathit{head}}() \\ \text{pre} &= \{\text{do}_{\mathit{deliver}}(), \text{level}(v), \mathit{at}(p, i), \mathit{goal}(p, g)\} \\ \text{eff} &= \{\neg\text{do}_{\mathit{deliver}}(), \text{do}_{\mathit{deliver\_all}_1}(p, i, g, v)\} \end{aligned}$$

$$\begin{aligned} \text{name} &= \mathit{deliver\_one}_1(p, i, g, v) \\ \text{pre} &= \{\text{do}_{\mathit{deliver\_one}_1}(p, i, g, v), \text{level}(v), \text{next}(v, w)\} \\ \text{eff} &= \{\neg\text{do}_{\mathit{deliver\_one}_1}(p, i, g, v), \neg\text{level}(v), \text{level}(w), \\ &\qquad \text{do}_{\mathit{pickup}}(p, i), \text{do}_{\mathit{deliver\_one}_2}(p, i, g, v)\} \end{aligned}$$

$$\begin{aligned} \text{name} &= \mathit{deliver\_one}_2(p, i, g, v) \\ \text{pre} &= \{\text{do}_{\mathit{deliver\_one}_2}(p, i, g, v), \text{level}(v), \text{next}(v, w)\} \\ \text{eff} &= \{\neg\text{do}_{\mathit{deliver\_one}_2}(p, i, g, v), \neg\text{level}(v), \text{level}(w), \\ &\qquad \text{do}_{\mathit{putdown}}(p, i), \text{do}_{\mathit{deliver\_all}_2}(p, i, g, v)\} \end{aligned}$$

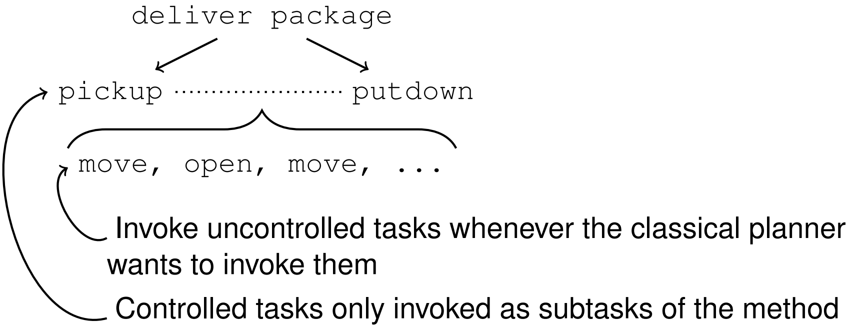
Method Translation

$$
\begin{aligned}
\text{name} &= \mathit{deliver\_one}_{\mathit{head}}() \\
\text{pre} &= \{\text{do}_{\mathit{deliver}}(), \text{level}(v), \mathit{at}(p, i), \mathit{goal}(p, g)\} \\
\text{eff} &= \{\neg\text{do}_{\mathit{deliver}}(), \text{do}_{\mathit{deliver\_all}_1}(p, i, g, v)\}
\end{aligned}
$$

$$
\begin{aligned}
\text{name} &= \mathit{deliver\_one}_1(p, i, g, v) \\
\text{pre} &= \{\text{do}_{\mathit{deliver\_one}_1}(p, i, g, v), \text{level}(v), \text{next}(v, w)\} \\
\text{eff} &= \{\neg\text{do}_{\mathit{deliver\_one}_1}(p, i, g, v), \neg\text{level}(v), \text{level}(w), \\
&\quad \text{do}_{\mathit{pickup}}(p, i), \text{do}_{\mathit{deliver\_one}_2}(p, i, g, v)\}
\end{aligned}
$$

$$
\begin{aligned}
\text{name} &= \mathit{deliver\_one}_2(p, i, g, v) \\
\text{pre} &= \{\text{do}_{\mathit{deliver\_one}_2}(p, i, g, v), \text{level}(v), \text{next}(v, w)\} \\
\text{eff} &= \{\neg\text{do}_{\mathit{deliver\_one}_2}(p, i, g, v), \neg\text{level}(v), \text{level}(w), \\
&\quad \text{do}_{\mathit{putdown}}(p, i), \text{do}_{\mathit{deliver\_all}_2}(p, i, g, v)\}
\end{aligned}
$$

Control Predicates



deliver package

pickup .... putdown

move, open, move, ...

Invoke uncontrolled tasks whenever the classical planner wants to invoke them

Controlled tasks only invoked as subtasks of the method

- How to achieve this? A *control predicate*
  - Put a new precondition into pickup and putdown
  - Assert this precondition in the PDDL translation of the method