HyperTensioN and Total-order Forward Decomposition Optimizations

JAAMAS Track AAMAS 2026, Paphos, Cyprus

M. C. Magnaguagno[1] **F. Meneguzzi**[1,2] L. de Silva[3]

[1]PUCRS, Porto Alegre, Brazil [2]University of Aberdeen, UK [3]University of Cambridge, UK

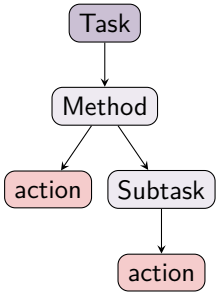
May 2026

Outline

Hierarchical Task Network Planning

- Tasks decomposed via methods (expert recipes)
- Methods encode *how* to achieve compound tasks
- Decomposition continues until only primitive actions remain
- HTN IPC 2020: new generation of planners

Key formalisms: SHOP2, JSHOP2, HDDL



Hierarchical Task Network Planning

- Tasks decomposed via methods (expert recipes)
- Methods encode *how* to achieve compound tasks
- Decomposition continues until only primitive actions remain
- HTN IPC 2020: new generation of planners

Key formalisms: SHOP2, JSHOP2, HDDL

HDDL domain (HTN)

```
(:task travel
 :parameters (?a ?b))

(:method m_direct
 :task (travel ?a ?b)
 :precondition (road ?a ?b)
 :ordered-subtasks
   (drive ?a ?b))

(:method m_via
 :task (travel ?a ?b)
 :precondition (road ?a ?m)
 :ordered-subtasks (and
   (travel ?a ?m)
   (drive ?m ?b)))
```

Hierarchical Task Network Planning

- Tasks decomposed via methods (expert recipes)
- Methods encode *how* to achieve compound tasks
- Decomposition continues until only primitive actions remain
- HTN IPC 2020: new generation of planners

Key formalisms: SHOP2, JSHOP2, HDDL

AgentSpeak (Jason)

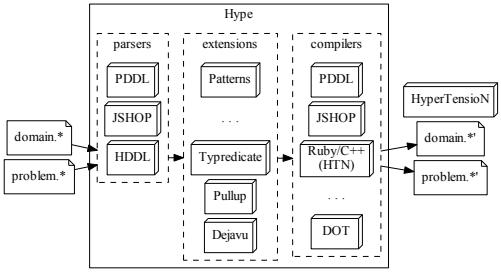
```
+!travel(A, B)
  : road(A, B)
  <- drive(A, B).

+!travel(A, B)
  : road(A, M)
  <- !travel(A, M);
     drive(M, B).
```

This should be familiar to AgentSpeak-style programmers

HyperTensioN: Three-stage Compiler Architecture

- Front-end: parse PDDL, HDDL, JSHOP inputs
- Middle-ends: transform/optimize domain IR
- Back-end: compile to Ruby/C++ planner code
- Lifted Total-order Forward Decomposition (TFD)
- No grounding — handles infinite object sets
- **Winner: HTN IPC 2020 total-order track**



Hype compiler + HyperTensioN TFD planner pipeline

Typredicate: Type-based Predicate Specialization

- When using type hierarchies, some variables unify with wrong-subtype objects at higher levels
- **Example (Transport):**
  - `(at ?obj - locatable ?l - location)`, this is a shared predicate
  - `drive` operator: `?obj` can be *vehicle* or *package*
  - Package substitutions are never valid for `drive`
- We solve this by splitting predicate per concrete subtype
  - `at` $\rightarrow$ `at_vehicle_location` / `at_package_location`
- Prunes substitution space; critical for Transport domain

Pullup: Precondition Propagation

- Operator preconditions tested too late in search
- If earlier steps cannot *bring about* p, then p must hold at method entry, so we test it sooner
- Our solution is fixed-point propagation up the HTN hierarchy
  - Removes dead branches before search begins
  - Replaces free variables with unique valid bindings
  - Discards impossible operator/method combinations
- Inspired by de Silva et al.'s *mentioned* predicates

Transport — before Pullup

```
; BEFORE Pullup
(:method m-load
 :parameters (?v ?l ?p ?s1 ?s2)
 :task (load ?v ?l ?p)
 :subtasks (pick-up ?v ?l ?p ?s1 ?s2))

(:action pick-up
 :parameters (?v ?l ?p ?s1 ?s2)
 :precondition (and
   (at ?v ?l) (at ?p ?l)
   (capacity-predecessor ?s1 ?s2)
   (capacity ?v ?s2))
 :effect (...))
```

Pullup: Precondition Propagation

- Operator preconditions tested too late in search
- If earlier steps cannot *bring about* p,
  then p must hold at method entry, so we test it sooner
- Our solution is fixed-point propagation up the HTN hierarchy
  - Removes dead branches before search begins
  - Replaces free variables with unique valid bindings
  - Discards impossible operator/method combinations
- Inspired by de Silva et al.'s *mentioned* predicates
  And other work from AAMAS
- Analogous to dead code elimination in compilers

Transport — before Pullup

```
; BEFORE Pullup
(:method m-load
 :parameters (?v ?l ?p ?s1 ?s2)
 :task (load ?v ?l ?p)
 :subtasks (pick-up ?v ?l ?p ?s1 ?s2))

(:action pick-up
 :parameters (?v ?l ?p ?s1 ?s2)
 :precondition (and
   (at ?v ?l) (at ?p ?l)
   (capacity-predecessor ?s1 ?s2)
   (capacity ?v ?s2))
 :effect (...))
```

Transport — after Typredicate + Pullup

```
; AFTER Typredicate + Pullup
(:method m-load ; preconditions from pick-up
 :parameters (?v ?l ?p ?s1 ?s2)
 :task (load ?v ?l ?p)
 :precondition (and
   (at_vehicle_location ?v ?l)
   (at_package_location ?p ?l)
   (capacity-predecessor ?s1 ?s2)
   (capacity ?v ?s2))
 :subtasks (pick-up ?v ?l ?p ?s1 ?s2))

(:action pick-up
 :parameters (?v ?l ?p ?s1 ?s2)
 :precondition ()   ; all pulled to m-load!
 :effect (...))
```

Dejavu: Cycle Detection for Recursive Domains

- Recursive methods can lead to infinite loops in TFD
- Domain expert workaround: manual `visited` predicates
- Our solution is automatic cycle detection via external cache
  - Adds invisible `visit`/`unvisit` primitive tasks
  - Marks which method+binding pairs are in progress
  - Cache persists across backtracking (state does not)
- Falls back to full state comparison when signatures unavailable
- Critical for Blocksworld-HPDDL, Hiking, Logistics, Transport, ...

HDDL (Rover)

```
; base: direct traverse
(:method m_direct
 :task (navigate2 ?x ?from ?to)
 :precondition
  (can_traverse ?x ?from ?to)
 :ordered-subtasks
  (navigate ?x ?from ?to))

; recursive: step via ?mid
(:method m_via
 :task (navigate2 ?x ?from ?to)
 :precondition
  (and (not (visited ?mid))
       (can_traverse ?x ?from ?mid))
 :ordered-subtasks (and
  (navigate ?x ?from ?mid)
  (visit ?mid)        ; mark
  (navigate2 ?x ?mid ?to)
  (unvisit ?mid)))    ; unmark

(:action visit
 :parameters (?p - waypoint)
 :precondition ()
 :effect (visited ?p))

(:action unvisit
 :parameters (?p - waypoint)
 :precondition (visited ?p)
 :effect (not (visited ?p)))
```

Dejavu: Cycle Detection for Recursive Domains

- Recursive methods can lead to infinite loops in TFD
- Domain expert workaround: manual `visited` predicates
- Our solution is automatic cycle detection via external cache
  - Adds invisible `visit`/`unvisit` primitive tasks
  - Marks which method+binding pairs are in progress
  - Cache persists across backtracking (state does not)
- Falls back to full state comparison when signatures unavailable
- Critical for Blocksworld-HPDDL, Hiking, Logistics, Transport, ...

AgentSpeak (Gold Miners)

```
// base case: already there
+!go_to(X,Y) : at(X,Y) <- true.

// recursive case: step via NX,NY
+!go_to(X,Y)
   : not visited(X,Y) &
     step(X,Y,NX,NY)
   <- +visited(X,Y);    // mark
      move(NX,NY);
      !go_to(X,Y);
      -visited(X,Y).    // unmark
```

Experimental Results — IPC 2020 Benchmark

24 domains · 892 instances · 16 GB RAM · 60 s timeout

Config	Solved
N (baseline)	370
T (Typredicate)	370
P (Pullup)	391
D (Dejavu)	491
TP	395
PD	540
TD	491
TPD	**555**

- D most impactful single extension (+121 over N)
- T alone ≈ N (useful only for Transport)
- PD and TPD best overall
- 7 domains unaffected by any extension
- **Won IPC 2020 with TPD**

Conclusions

- HyperTensioN: three-stage compiler for modular HTN optimizations
- Typredicate — type specialization prunes substitution space
- Pullup — precondition propagation eliminates dead branches early
- Dejavu — external cache enables TFD on recursive domains
- TPD configuration **won HTN IPC 2020 total-order track**
- Optimizations are *domain transformations*:
  applicable to other total-order HTN planners

Full evaluation: *Magnaguagno et al., JAAMAS 2025*

Thank You

Questions?

M. C. Magnaguagno <mauricio.magnaguagno@acad.pucrs.br>
F. Meneguzzi <felipe.meneguzzi@abdn.ac.uk>
L. de Silva <lavindra.desilva@eng.cam.ac.uk>

https://ipc2020.hierarchical-task.net/