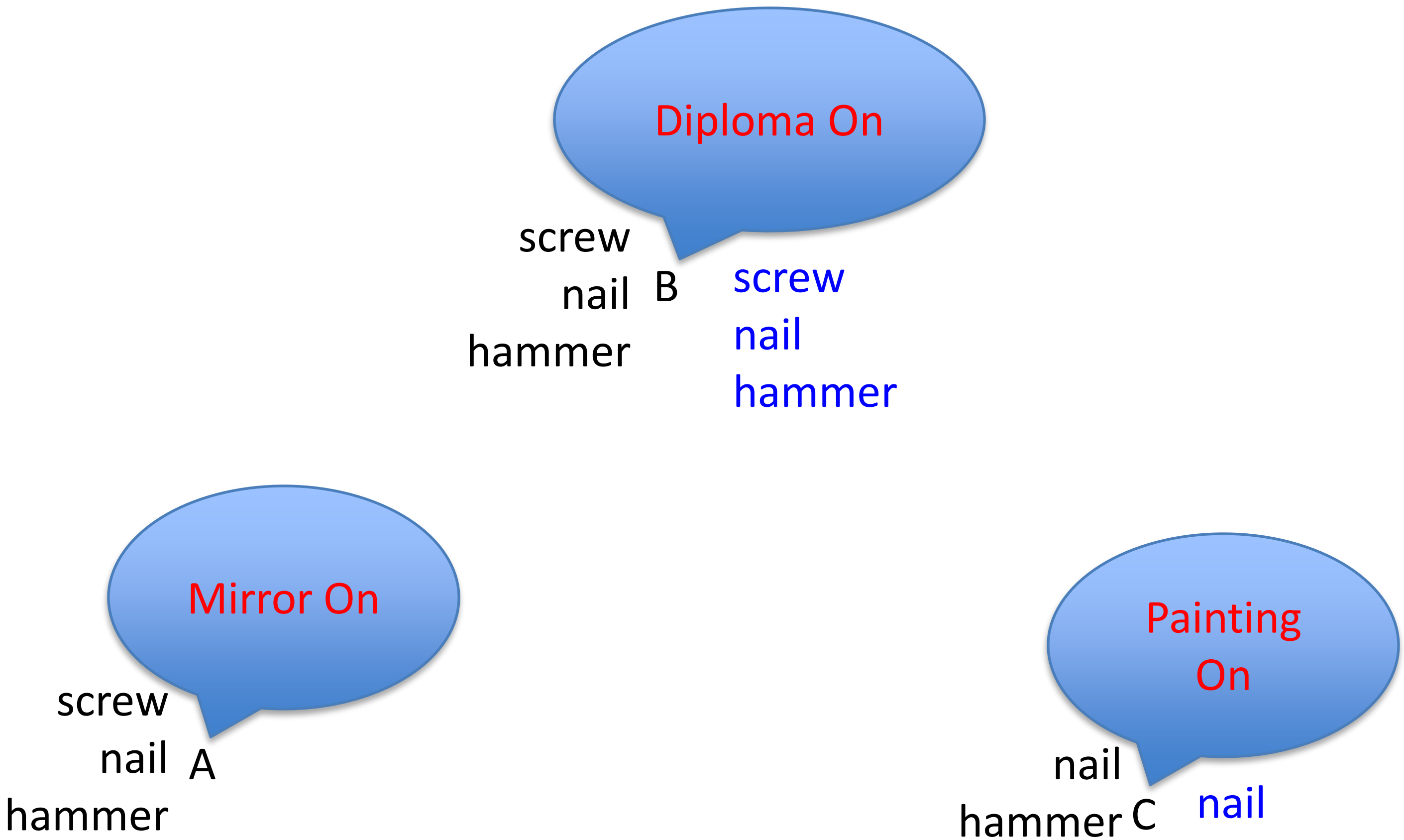
Reasoning and Planning with Cooperative Actions for Multi-agents Using Answer Set Programming

Tran Cao Son (New Mexico State University)
Chiaki Sakama (Wakayama University)

Outline

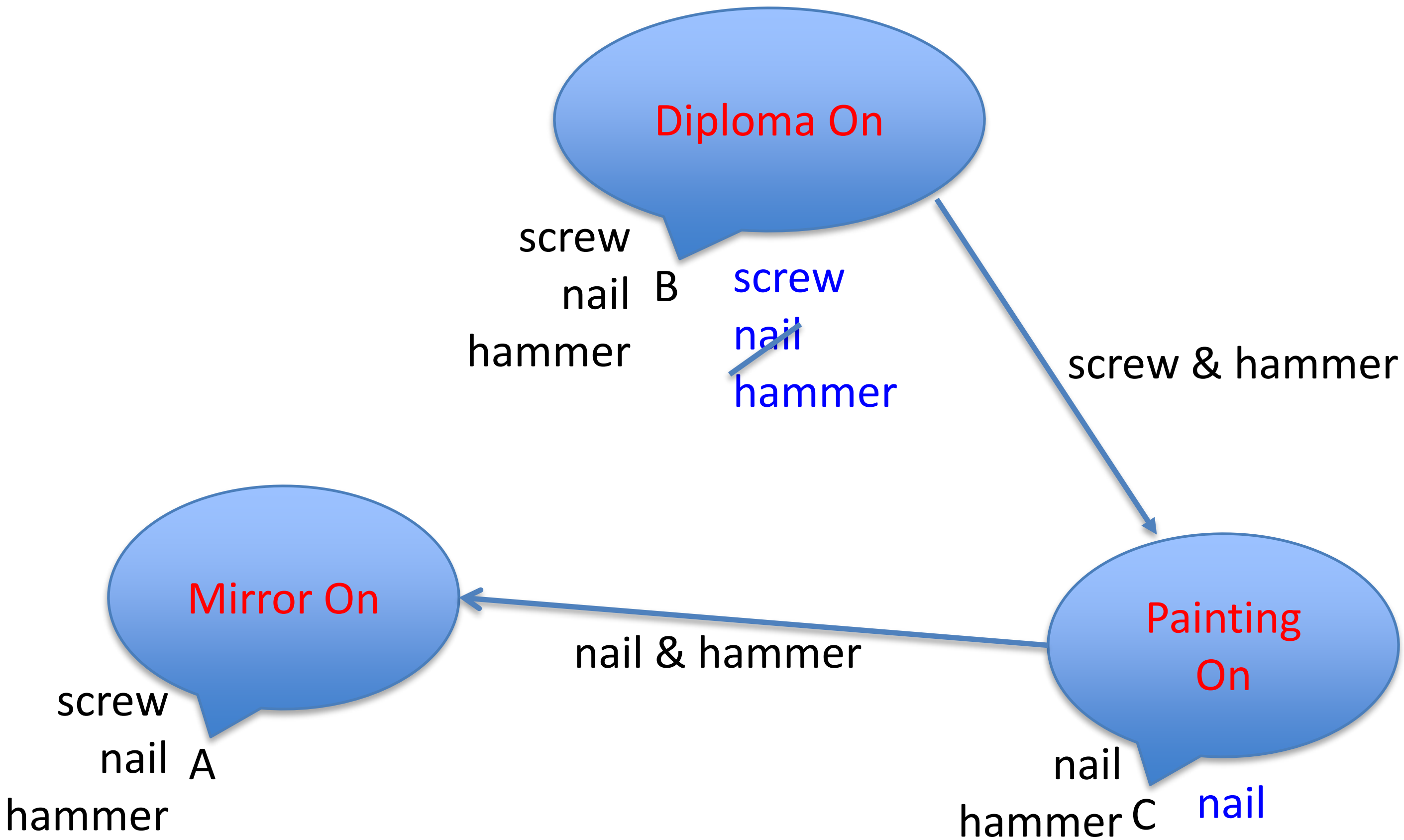
- Motivation
- An action language with cooperative actions
- Multi-agent planning with cooperative actions
- Discussion
- Conclusion

Motivation - Sharing



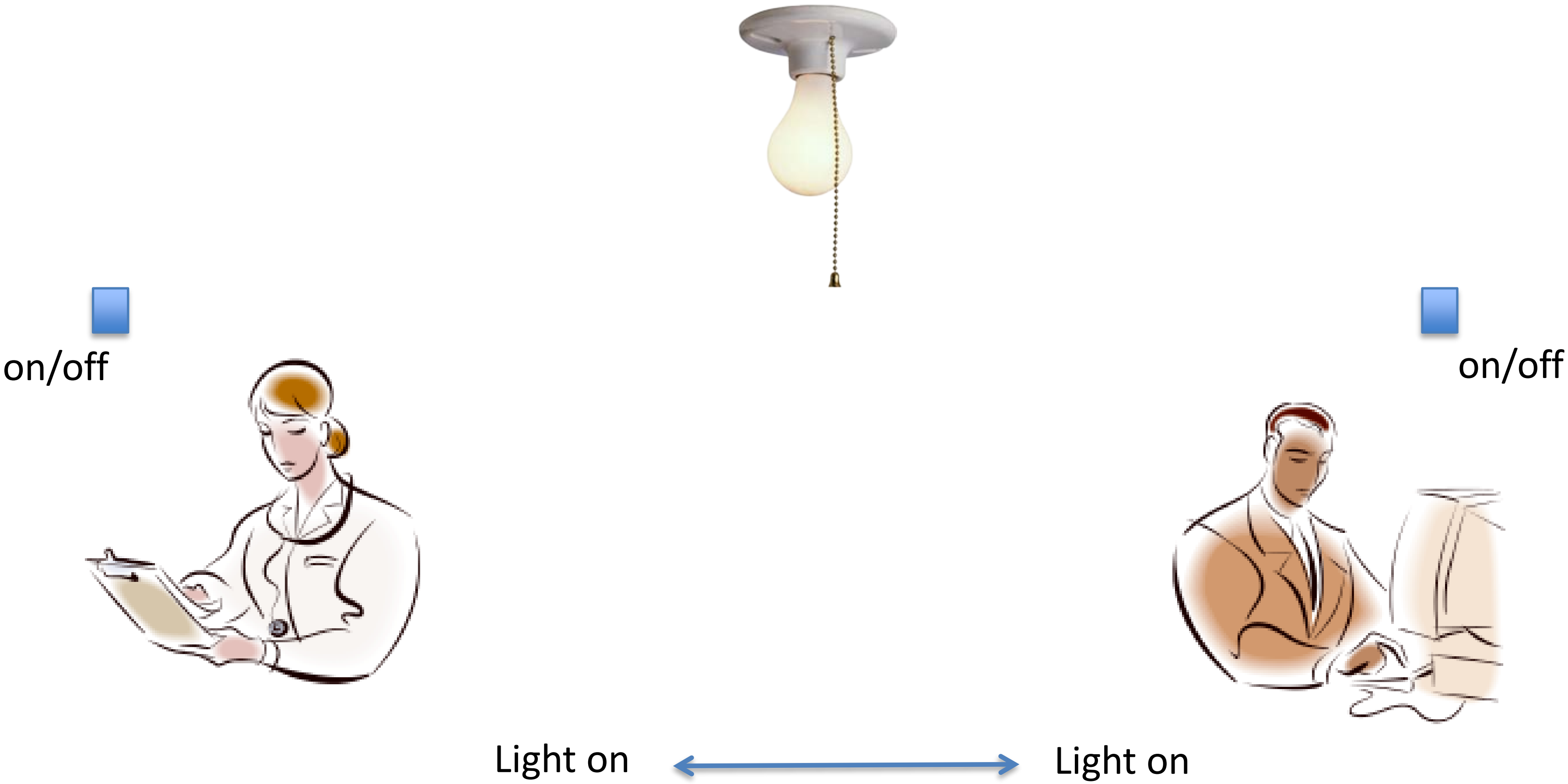
Can all three achieve their goals? Yes, but they need to cooperate!

Motivation - Sharing



Can all three achieve their goals? Yes, but they need to cooperate!

Motivation - Interference



Cooperative actions

- are those that establish/impose some conditions for/on others
- might or might not affect the local world of the action executor as well as other agent’s world
- are needed in multi-agent planning

Overall questions

- Given: a multi-agent system with cooperative actions
- Questions:
  - how to represent and reason with cooperative actions?
  - can *all* agents achieve their goals, if so how?

An action language with cooperative actions

- Extension of language A [Gelfond & Lifschitz, 93] with *cooperative actions*
- Transition function based semantics
- Each agent knows
  - knows about its actions
  - knows who can help with their need
  - knows who she can help

Specifying individual actions

- Effects of actions: *action* **causes** *effects* **if** *conditions*

  hang_with_nail **causes** *on_wall, -nail*

  (the object on the wall after the execution of *hang_with_nail* depends on the person, A: mirror, B: diploma, C: paiting)

- Executability condtions: *action* **executable** *conditions*

  hang_with_nail **executable** *has_nail, has_hammer*

Specifying a request

action ***requests*** *something* ***from*** *someone*
may_cause *effects* **if** *exec_conditions*

give_me_hammer **requets** *has_hammer*
from {B,C} **may_cause** *has_hammer*
if –*has_hammer*

Specifying a support

action ***provides*** *something* ***to*** *someone*

causes *effects* **if** *exec_conditions*

something: effects on the world of the agent requesting it

effects: effects on the world of the agent executing the action

has_this_hammer ***provides*** *has_hammer*

to {B,C} **causes -***has_hammer* **if** *has_hammer*

Domain with cooperative actions

- D = (DI, DC, I)
  - DI: individual (non-cooperative) actions
  - DC: cooperative actions
  - I: initial state
- Semantics
  - transition function based: defining a function φ for computing the successor state

Semantics – Non-cooperative actions

$$\varphi(a,s) = s + e^{+}(a,s) - e^{-}(a,s)$$

- $e^{+}(a,s)$ is the positive effects of a in s
- $e^{-}(a,s)$ is the negative effects of a in s

Semantics – Cooperative actions

- Request: *r requests Υ from i* **may_cause** ψ **if ω**
  - Executable only if ω is satisfied
  - Successful: $\varphi(r(\Upsilon,i), s) = s - \psi^* + \psi$ where $\psi^*=\{-l \mid l \text{ in } \psi\}$ (complement of ψ)
  - Unsuccessful $\varphi(r(\Upsilon,i), s) = s$
- Provides: *p provides Υ to i* **causes** ψ **if** ω
  - Executable only if ω is satisfied
  - Successful: $\varphi(p(\Upsilon,i), s) = s - \psi^* + \psi$

φ – is a nondeterministic function

Reasoning with cooperative actions

- D = (DI, DC, I)
  - DI: individual actions
  - DC: cooperative actions
  - I: initial state
- Trajectory: $\alpha=[s_0\ a_1\ s_1\ \ldots\ s_{n-1}\ a_n\ s_n]$

  s_{i+1} belongs to $\varphi(a_i, s_i)$

  $a_j = r(\Upsilon, i)$ is satisfied in α if $s_{j-1} \neq s_j$
- ϕ is true after α if ϕ holds in s_n

Planning with cooperative actions

- P = (DI, DC, I, G)
  - DI: individual actions
  - DC: cooperative actions
  - I: initial state
  - G: goal
- Possible plans:
  - a trajectory $\alpha=[s_0\ a_1\ s_1\ \ldots\ s_{n-1}\ a_n\ s_n]$
  - s_0 satisfies the initial conditions I
  - s_n satisfies goal G

Multi-agent system

- Each agent has its own (local) domain description
- Agents might have different/same representation
  - (A, B, has_hammer, has_hammer)
- Execution of an action might change local/global world
  - A turn the light on → light will be on for B
- Agents might request for help
- Conditions on joint-actions (parallel, non-parallel)
  - {(A, lift_move_table), (B, lift_move_table)}
  - {(A, turn_on), (B, turn_on)}

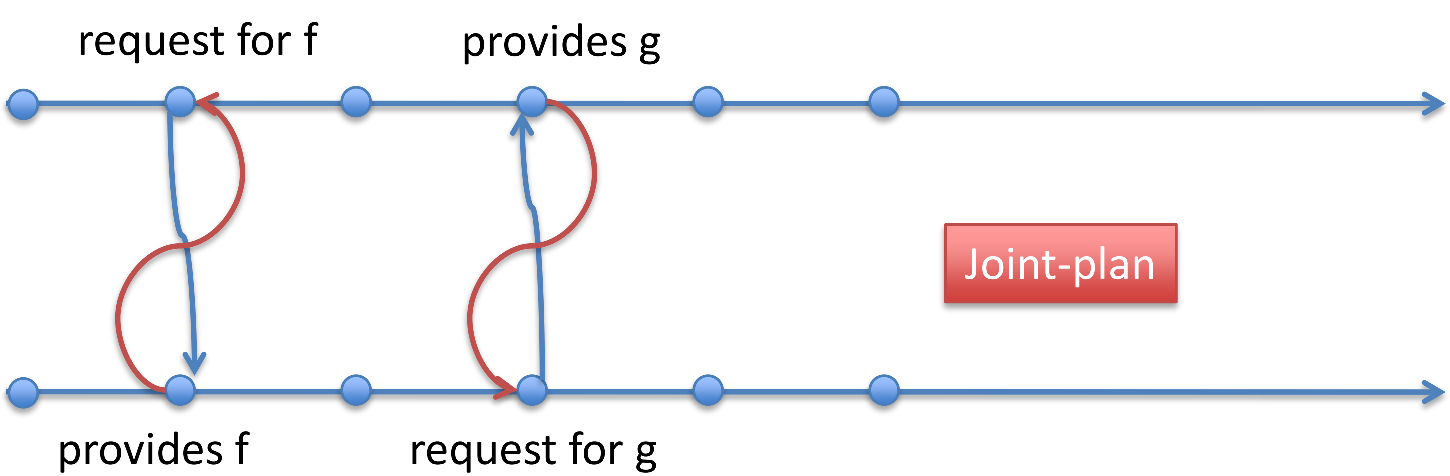
Multi-agent planning problem

$<Ag, P_{Ag}, F, IC, C>$

- Ag: set of agents
- P_{Ag}: set of planning problem, one for each agent
- F: set of interacting fluents between agents
- IC: set of pairs of agents and actions that cannot be executed concurrently
- C: set of actions that have to be executed concurrently

Solution of <Ag, P_{Ag},F, IC, C>

- Requirements:
  - consists of possible plans for agents in Ag
  - for each request-action, which is assumed to be satisfied by some agent, there exists some agent who provides for the request
  - satisfies IC and C



Computing joint-plan <Ag, P_{Ag},F, IC, C> using Answer Set Programming

- ASP
  - logic programming under answer set semantics
  - simple syntax, expressive
  - available solvers (active development by ASP community)
- Computing joint-plan using ASP
  - represent a multi-agent planning problem as logic programs
  - compute answer sets
  - extract plans

Computing joint-plan <Ag, P_{Ag},F, IC, C> using Answer Set Programming

- Translating each planning problem P_i into a program $\pi(P_i)$ such that each answer set of $\pi(P_i)$ is a possible solution of $\pi(P_i)$
- Combining answer sets of P_i to create joint-plan: joint-plans equivalent to "compatible answer sets"

P_i = (DI,DC,I,G) and $\pi_i(P_i)$

- Rules representing effects of non-cooperative actions:
  (a **causes** f **if** pre) in DI
  h(f, t+1) :- o(a,t), executable(a,t), h(pre,t)
- Rules representing effects of cooperative actions:
  (r **request** γ **from** i **may_cause** ξ **if** Φ) in DC
  0 {ok(r(γ,i), t+1)} 1 :- o(r(γ,i), t)
  h(ξ,T) :- ok(r(γ,i), t+1)
  (p **provides** γ **to** i **causes** ξ **if** Φ) – same as for normal actions
- Rules representing initial state
  h(f,0) if f belongs to I
- Rules representing goal
  :- not h(f,n) if g in G
- Rules generating action occurrences
  1 {occ(a,T) : action(a)} :- T < n
- Inertial rules
  h(f, T+1) :- h(f, T), not h(-f, T+1)
  h(-f, T+1) :- h(-f, T), not h(f, T+1)

Computing joint-plan <Ag, P_{Ag},F, IC, C> using Answer Set Programming

- Distributed computation: Combining answer sets of $\pi(P_i)$ to create joint-plan: joint-plans equivalent to "compatible answer sets"
- Centralized computation:
  - Combining $\pi(P_i)$ into a single program π
  - Adding to $\pi(P_i)$ constraints expressing the constrains of the problem π
  - Computing answer set of π

Related works

- Most works in multiagent planning
  - employ partial plan representation
  - assume that the partial plans exist, reasoning about effects of actions
  - concentrate on synchronizing the partial plans so that constraints can be satisfied
  - less focus on reasoning about effects of actions
- Our approach
  - focus on reasoning about effects of actions
  - use standard approach to compute joint-plans
  - can be extended to allow different types of actions (non-deterministic actions, parallel actions, etc.)

Conclusions and future works

- In this paper:
  - Framework for reasoning and planning with cooperative actions
  - Implementation in answer set programming
- Future works:
  - Experimenting with different solvers
  - Investigating algorithms for distributed computation of compatible answer sets
  - Combining negotiation within planning