Commonsense Reasoning for Agents

- Abstract Game Design
- Event Calculus
- Reasoning Modes: Abduction vs. Deduction
- Using Event Calculus: Hierarchical Planning, Agent Control, Negotiation

The Game-Design Process

- Abstract game mechanics
  - State and state evolution
- Concrete game representation
  - Audiovisual representation of the abstract mechanics
- Thematic content
  - Real-world reference of the game
- Control mapping
  - How a player interacts with a game

Decomposing a Game

- Consider games on two levels: abstract and concrete
- The abstract game is thematic, with high-level dynamics
  - Bundles thematic elements and high-level game dynamics
  - Person avoiding a car
  - Duck avoiding a bullet
  - Mechanic pumping up a tire
- The concrete game is a templated bundle of executable code
  - Bundles dynamics, representation, control mapping
  - First-person crosshairs, player wins if shoots [Sprite-A]
  - 2d side view, player plays [Sprite-A] and wins if avoids [Sprite-B] for 5sec
- Generation problem: Generate games that “make sense“ and plug them into concrete game

Reasoning About Mechanics

- Thematic reasoning system procedurally encodes mechanics
  - Therefore, difficult to reason about mechanics
- Event Calculus to reason about mechanics and state representation
- Two applications
  - Game design assistant for expert
  - Automated design discovery

The Language of the Event Calculus

An event calculus signature contains

- Function symbols into sorts FLUENT, EVENT, and TIME
- *HoldsAt* : $(\neg)$FLUENT $\times$ TIME
- *Happens*: EVENT $\times$ TIME
- *Initiates*: EVENT $\times$ FLUENT $\times$ TIME
- *Terminates*: EVENT $\times$ FLUENT $\times$ TIME
- *Initially*: $(\neg)$FLUENT

Applying Event Calculus to Games

- Fluents encode game state
- Events more general than actions: encode player input (button press) and game-state initiated events (collision, player wins, ...)
- Game mechanics are state-evolution rules, i.e. the changes in the game state are caused by events, and the events are triggered by game state
  - Colliding with an enemy (event) causes player to lose health (fluent value change)
  - Reaching 100 points (state trigger) causes player to win game (event)

Example: Simple Tile World

- Sprites occupying the same tile location collide

  $sprite1 \neq sprite2 \wedge$
  $HoldsAt(At(sprite1,x,y),t) \wedge$
  $HoldsAt(At(sprite2,x,y),t) \supset$
  $Happens(Collide(sprite1,sprite2),t)$

- Inventory is empty at start of the game; *gain* and *lose* events change inventory; used items must be in the inventory
  - $Initially(\neg InInventory(x))$
  - $Initiates(Gain(x),InInventory(x),t)$
  - $Terminates(Lose(x),InInventory(x),t)$
  - $Happens(UseOn(x,object),t) \supset HoldsAt(InInventory(x),t)$

- Gain event caused by colliding with item that can be picked up

  $Happens(Collide(sprite1,sprite2),t) \wedge HoldsAt(HasSprite(sprite1,Player),t) \wedge$
  $HoldsAt(HasSprite(sprite2,x),t) \wedge HoldsAt(PickUpable(x),t) \supset$
  $Happens(Gain(x),t)$

Foundational Axioms

1. $Happens(e,t_1) \wedge t_1<t_2 \wedge Initiates(e,f,t_1) \wedge \neg Clipped(t_1,f,t_2) \supset HoldsAt(f,t_2)$
2. $Initially(f) \wedge 0 \leq t \wedge \neg Clipped(0,f,t) \supset HoldsAt(f,t)$
3. $Happens(e,t) \wedge Terminates(e,f,t) \wedge t_1 \leq t<t_2 \supset Clipped(t_1,f,t_2)$

4. $Happens(e,t_1) \wedge t_1<t_2 \wedge Terminates(e,f,t_1) \wedge \neg Declipped(t_1,f,t_2) \supset HoldsAt(\neg f,t_2)$
5. $Initially(\neg f) \wedge 0 \leq t \wedge \neg Declipped(0,f,t) \supset HoldsAt(\neg f,t)$
6. $Happens(e,t) \wedge Initiates(e,f,t) \wedge t_1 \leq t<t_2 \supset Declipped(t_1,f,t_2)$

Reasoning Modes

- Deduction = Forward simulation
  - Forward simulate a game just like you would with procedural code
- Abduction = Planning
  - Ask questions about reachable game states, about ways of overcoming challenges, etc.
- Induction = Player modeling
  - Take concrete player traces (either observed or abduced) and build player models

Abduction

Abduction: Definition

Given

- Normal logic program P
- Set of abducible predicates A
- Integrity constraints IC

A solution to a goal G is a set of ground atoms Δ from A such that

- G is entailed by $P \cup \Delta$ (including negation-as-failure principle)
- $P \cup \Delta \cup IC$ is consistent (i.e., does not entail *false*)

Abduction with Event Calculus

- Abducible *Happens*
- Domain-independent integrity constraint

$$HoldsAt(f,t) \wedge HoldsAt(\neg f,t) \supset false$$

- Answers to goals $HoldsAt(f_1,t_1) \wedge \ ... \wedge HoldsAt(f_n,t_n)$ are *plans*

Example

- Game rules $Initially(\neg Has(Money))$
  $x \neq Money \supset Initiates(Buy(x),Has(x),t)$
  $Initiates(Borrow(x),Has(x),t)$
  $Terminates(Buy(x),Has(Money),t)$
  $Happens(Buy(x),t) \supset HoldsAt(Has(Money),t)$
  $Happens(Buy(x),t) \supset 9 \leq t \leq 21$
- Goal $HoldsAt(Has(Key),15)$
- Example solutions
  - $\{Happens(Borrow(Key),12)\}$
  - $\{Happens(Borrow(Money),8),Happens(Buy(Key),9)\}$
- In practice, instead of “guessing“ ground instances, constraints are abduced, resulting in partial-order plans (including time constraints)

Hierarchical Planning

Example: Primitive Actions of Indoor Robot

- Fluents and primtive actions

Symbol	Type
In	ROOM $\mapsto$ FLUENT
In	PACK $\times$ ROOM $\mapsto$ FLUENT
Got	PACK $\mapsto$ FLUENT
Pick	PACK $\mapsto$ ACTION
Put	PACK $\mapsto$ ACTION
Gothrough	DOOR $\mapsto$ ACTION

- Axioms $HoldsAt(In(r),t) \wedge HoldsAt(In(p,r),t) \supset Initiates(Pick(p),Got(p),t)$
  $HoldsAt(In(r),t) \wedge HoldsAt(Got(p),t) \supset Initiates(Put(p),In(p,r),t)$
  $HoldsAt(In(r_1),t) \wedge Connects(d,r_1,r_2) \supset Initiates(Gothrough(d),In(r_2),t)$
  $HoldsAt(In(r),t) \supset Terminates(Gothrough(d),In(r),t)$

Compound Action (1)

- Definition of a compound action

$$Happens(GoToRoom(r,r),t,t)$$

$$Connects(d,r_1,r_2) \wedge Happens(Gothrough(d),t_1) \wedge$$
$$\neg Clipped(t_1,In(r_2),t_2) \wedge Happens(GoToRoom(r_2,r_3),t_2,t_3) \supset$$
$$Happens(GoToRoom(r_1,r_3),t_1,t_3)$$

- Effects of the compound action

$$HoldsAt(In(r_1),t) \supset Initiates(GoToRoom(r_1,r_2),In(r_2),t)$$
$$HoldsAt(In(r_1),t) \wedge r_1 \neq r_2 \supset Terminates(GoToRoom(r_1,r_2),In(r_1),t)$$

Compound Action (2)

- Definition of a compound action

$$Happens(GoToRoom(r_1,r_2),t_1,t_2) \wedge \neg Clipped(t_2,In(r_2),t_3) \wedge$$
$$\neg Clipped(t_1,In(p,r_2),t_3) \wedge Happens(Pick(p),t_3) \wedge$$
$$Happens(GoToRoom(r_2,r_3),t_4,t_5) \wedge \neg Clipped(t_3,Got(p),t_6) \wedge \neg Clipped(t_5,In(r_3),t_6) \wedge$$
$$Happens(Put(p),t_6) \wedge t_2<t_3<t_4 \wedge t_5<t_6 \supset$$
$$Happens(ShiftPack(p,r_1,r_2,r_3),t_1,t_6)$$

- Effects of the compound action

$$HoldsAt(In(r_1),t) \wedge HoldsAt(In(p,r_2),t) \supset Initiates(ShiftPack(p,r_1,r_2,r_3),In(p,r_3),t)$$
$$HoldsAt(In(r_1),t) \wedge HoldsAt(In(p,r_2),t) \supset Terminates(ShiftPack(p,r_1,r_2,r_3),In(p,r_2),t)$$

- Note: these effects follow from effects of primitive actions, which can be verified formally or by “inspection“

Hierarchical Planning

Hierarchical planning in the agent allows

- to build heuristics and expertise into planning
- to generate actions in progressive order (first action first)

Progressive planning fits well within an agent control cycle:

- A partial plan can be executed and give useful results
- Observe effects of actions and state of the environment to decide whether it is worth continuing with that plan

Agent Control Cycle

Agent Cycle


start
Observe
What is happening?
Reflect
What does it mean?
Plan
What do we want to change?
Act
What are we doing?

Agent Cycle

To cycle at time t:

- Record any observations at time t
- Select actions (events) to be executed in the interval $[t\,,\ t+\Delta t\,)$
- Attempt to perform these actions
- Record success or failure of the performed actions as observations
- Cycle at time $t+\Delta t$

Adding Actions and Observations

1. $Observed(f,t_1) \wedge t_1 \leq t_2 \wedge \neg Clipped(t_1,f,t_2) \supset HoldsAt(f,t_2)$
2. $Observed(\neg f,t_1) \wedge t_1 \leq t_2 \wedge \neg Declipped(t_1,f,t_2) \supset HoldsAt(\neg f,t_2)$

3. $Executed(e,t) \supset Happens(e,t)$
4. $SelectAction(e,t) \supset Happens(e,t)$

5. $Observed(\neg f,t) \wedge t_1 \leq t < t_2 \supset Clipped(t_1,f,t_2)$
6. $Observed(f,t) \wedge t_1 \leq t < t_2 \supset Declipped(t_1,f,t_2)$

- Now *SelectAction* is the abducible

Agent Negotiation

Communication between Agents

- Dialogues between agents as a means of interaction
- Usually based on fixed interaction protocols
- Negotiation is the process by which agents communicate with each other with the aim to come to a mutual agreement on some matter
- Example: negotiating for resource sharing and allocation

Examples

$Agent_1$ has a plan requiring resources A,B. It has A,E and is missing B.
$Agent_2$ has a plan requiring resources D,E. It has B,C,D and is missing E.

- $Agent_1$: Can you give me B?
- $Agent_2$: Yes, if you give me E.

- $Agent_1$: Can you give me B?
- $Agent_2$: No. Why do you want B?
- $Agent_1$: I have a goal G and plan ... for which I need B.
- $Agent_2$: You can achieve G with plan ... which needs C instead of B, and I can give you C if you give me E.

Communication Language

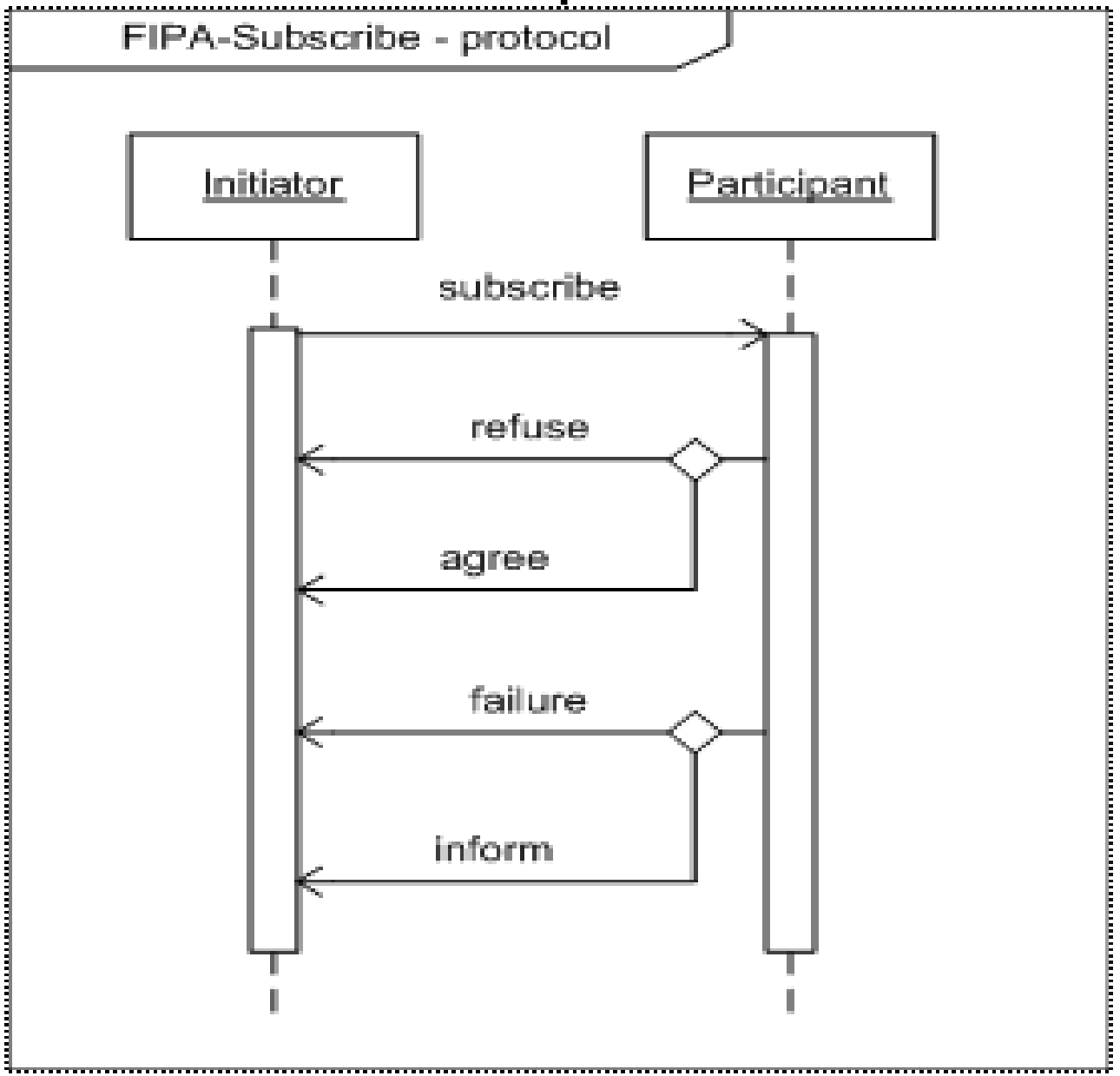
Tell(ag_1,ag_2,*content*)

content can be:

- request *Request(Give(r))* *Request(Give(r),*T_{start},T_{end}*)*
- accept
- reject
- ...

Content and form of dialogues are defined by interaction protocols.

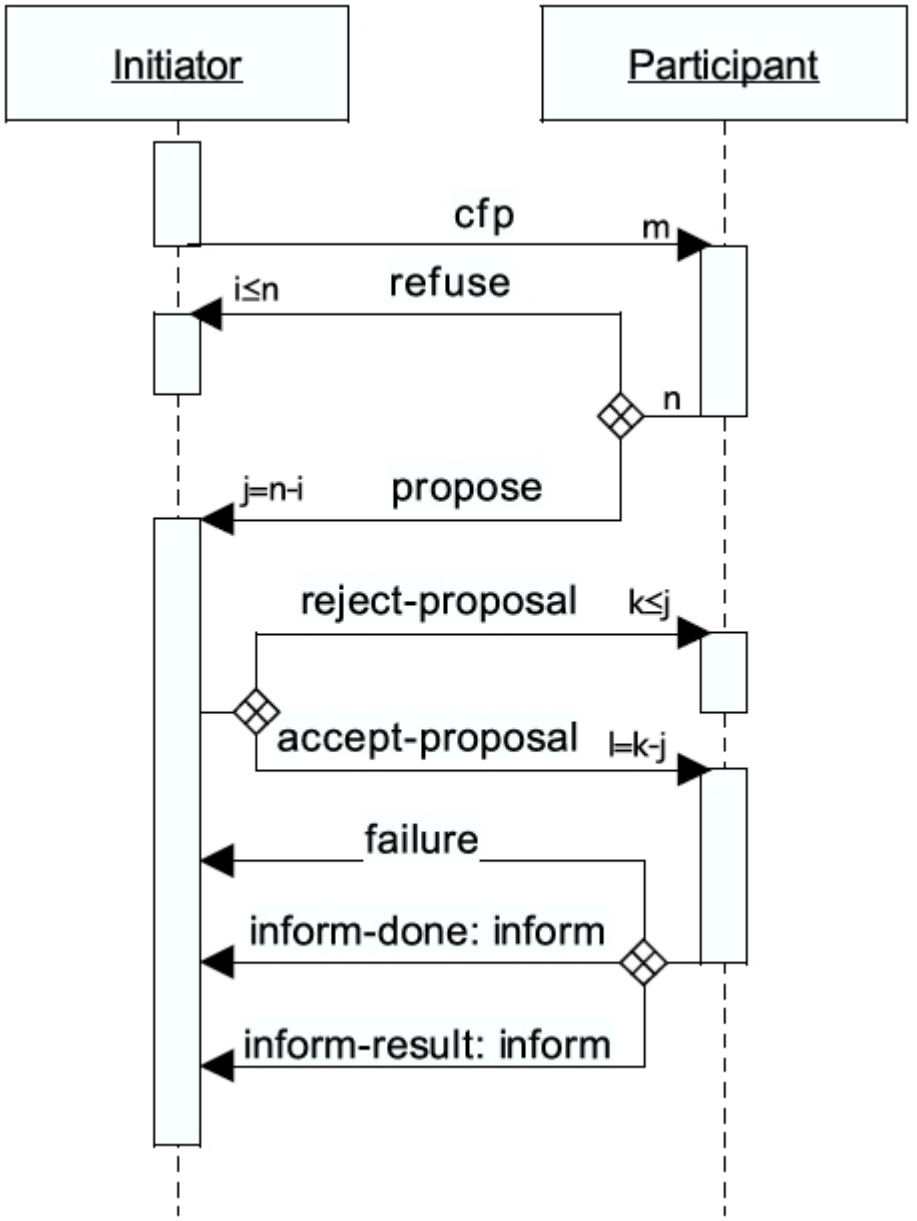
A Simple Interaction Protocol



FIPA defines standardized interaction protocols

FIPA = IEEE Foundation for Intelligent Physical Agent

Another Example: English Auction



Programming Interaction Policies for Agents

- Interaction policy expressed as constraint of the form

$$Message_i \wedge Condition \supset Message_{i+1}$$

- Intended meaning: If the agent receives $Message_i$ and *Condition* is satisfied, then it generates $Message_{i+1}$
- $Observed(Tell(c,Agent,Request(r)),t_1) \wedge HoldsAt(Have(r),t_1) \wedge HoldsAt(\neg Need(r),t_1) \supset Happens(Tell(Agent,c,Accept),t_2) \wedge t_1+5>t_2>t_1$

  $Observed(Tell(c,Agent,Request(r)),t_1) \wedge HoldsAt(\neg Have(r),t_1) \supset Happens(Tell(Agent,c,Reject),t_2) \wedge t_1+5>t_2>t_1$

  $Oberved(Tell(c,Agent,Request(r)),t_1) \wedge HoldsAt(Need(r),t_1) \supset Happens(Tell(Agent,c,Reject),t_2) \wedge t_1+5>t_2>t_1$