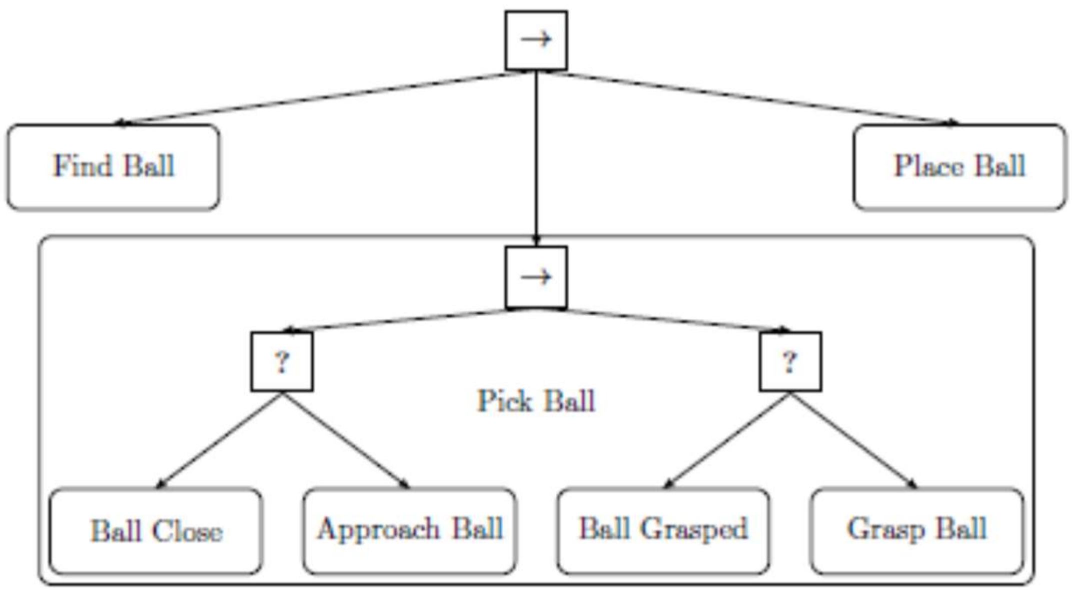
Behavior Trees

A Behavior Tree (BT) is a way to structure the switching between different tasks in an autonomous agent, such as a robot or a virtual entity in a computer game



M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

Behavior Trees

Developed in the computer game industry to increase modularity in the control structures of Non-Player Characters (NPCs)

At Carnegie Mellon University, BTs have been used extensively to do robotic manipulation

FSMs have long been the standard choice when designing a task switching structure, but they lack of modularity and flexibility

M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

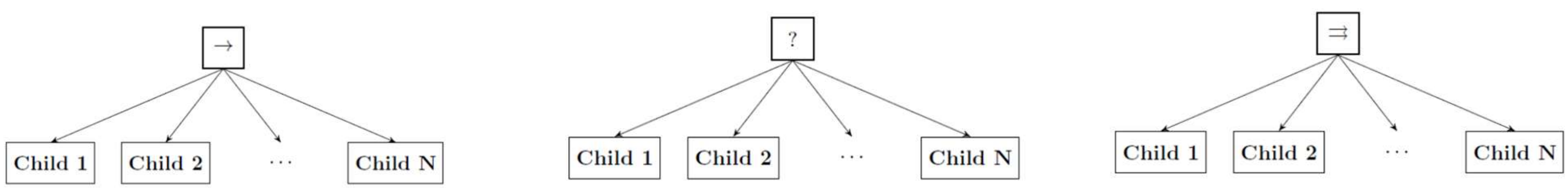
Behavior Trees

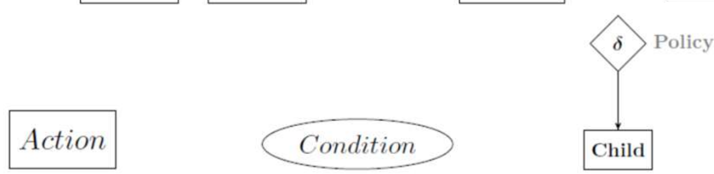
The execution starts from the root which sends enabling signals (ticks) that allows the execution of a child

- A node is executed if and only if it receives ticks.
- The child immediately returns **Running** to the parent, if its execution is under way, **Success** if it has achieved its goal, or **Failure** otherwise.

Nodes classified as **root**, **control flow** nodes, or **execution** nodes:

- Control flow nodes (Sequence, Fallback, Parallel, and Decorator)
- Execution nodes (Action and Condition)





(a) Action node. The label describes the action performed.
(b) Condition node. The label describes the condition verified.
(c) Decorator node. The label describes the user defined policy.

M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

Behavior Trees

The execution starts from the root which sends enabling signals (ticks) that allows the execution of a child

Nodes classified as **root**, **control flow** nodes, or **execution** nodes:

- Control flow nodes (Sequence, Fallback, Parallel, and Decorator)
- Execution nodes (Action and Condition)

Node type	Symbol	Succeeds	Fails	Running
Fallback	?	If one child succeeds	If all children fail	If one child returns Running
Sequence	$\rightarrow$	If all children succeed	If one child fails	If one child returns Running
Parallel	$\rightrightarrows$	If $\geq M$ children succeed	If $> N - M$ children fail	else
Action	text	Upon completion	If impossible to complete	During completion
Condition	text	If true	If false	Never
Decorator	$\Diamond$	Custom	Custom	Custom

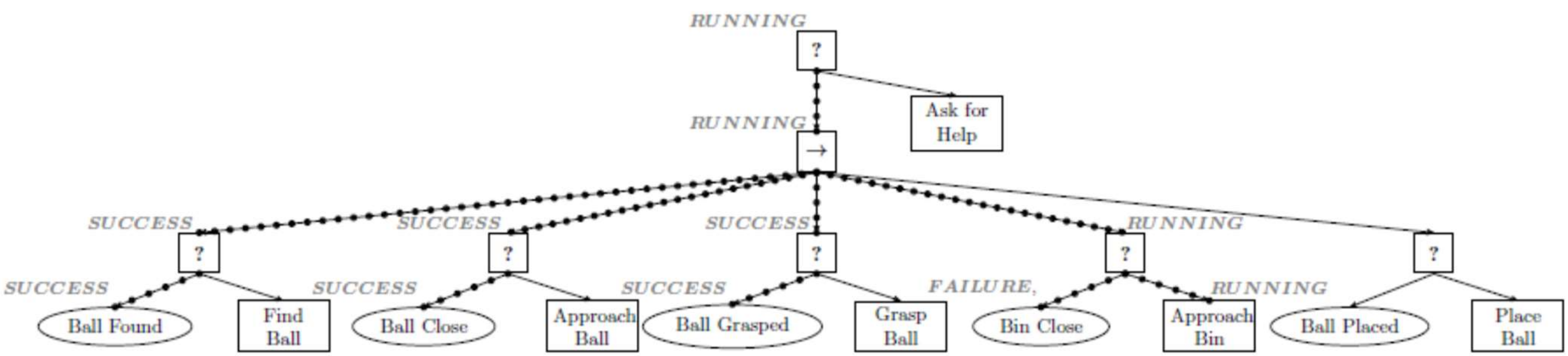
M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

Behavior Trees

The execution starts from the root which sends enabling signals (ticks) that allows the execution of a child

Nodes classified as **root**, **control flow** nodes, or **execution** nodes:

- Control flow nodes (Sequence, Fallback, Parallel, and Decorator)
- Execution nodes (Action and Condition)

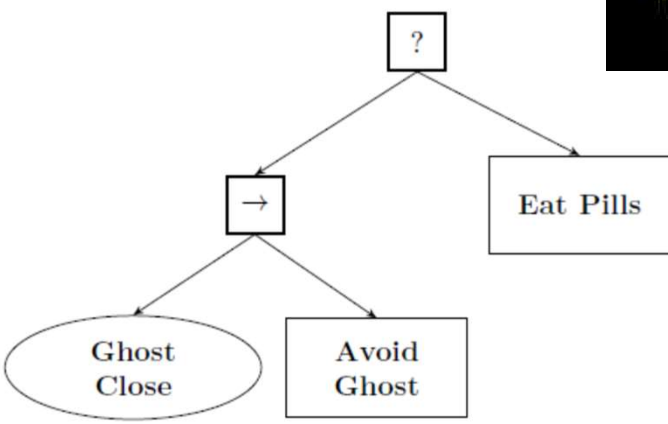


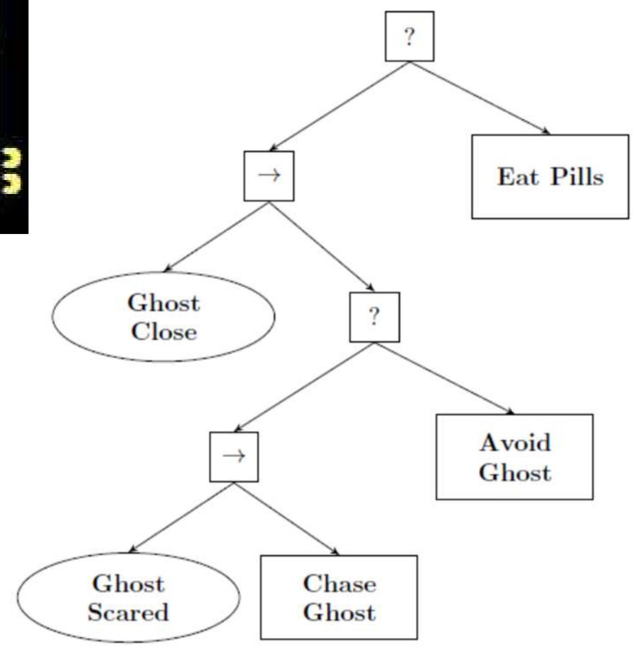
M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

Behavior Trees

The execution starts from the root which sends enabling signals (ticks) that allows the execution of a child

Pack-man example

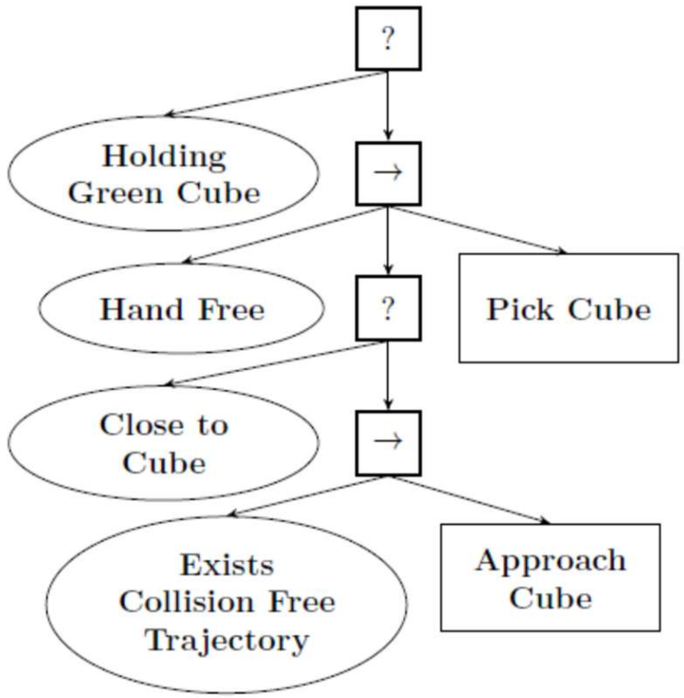


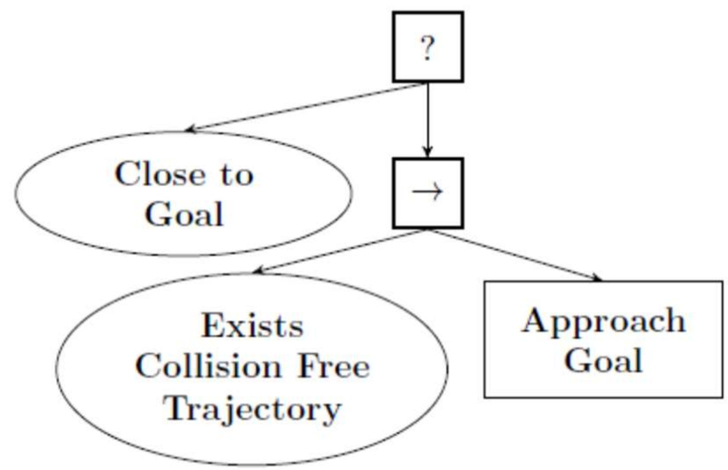


M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

Behavior Trees

Pick and Place scenario in CoppeliaSim https://btirai.github.io/

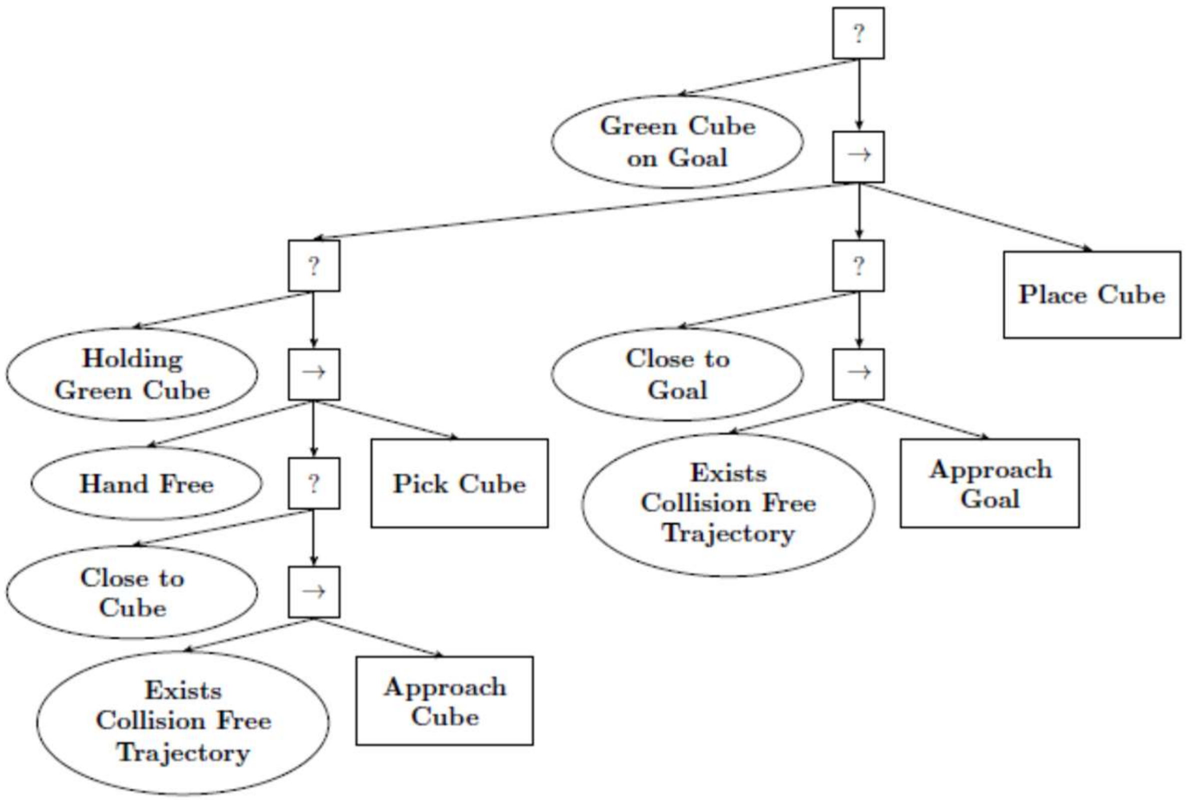




M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

Behavior Trees

The execution starts from the root which sends enabling signals (ticks) that allows the execution of a child



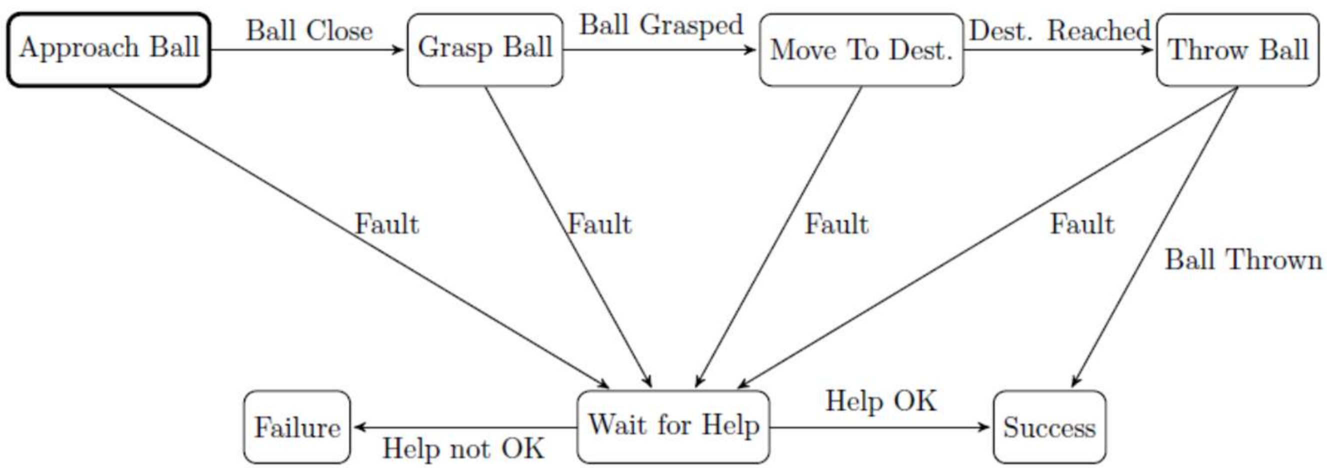
M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

Behavior Trees

Amazon Picking Competition
The Amazon Robotics/Picking Challenge http://amazonpickingchallenge.org/

M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

BTs vs FSMs



Finite State Machines

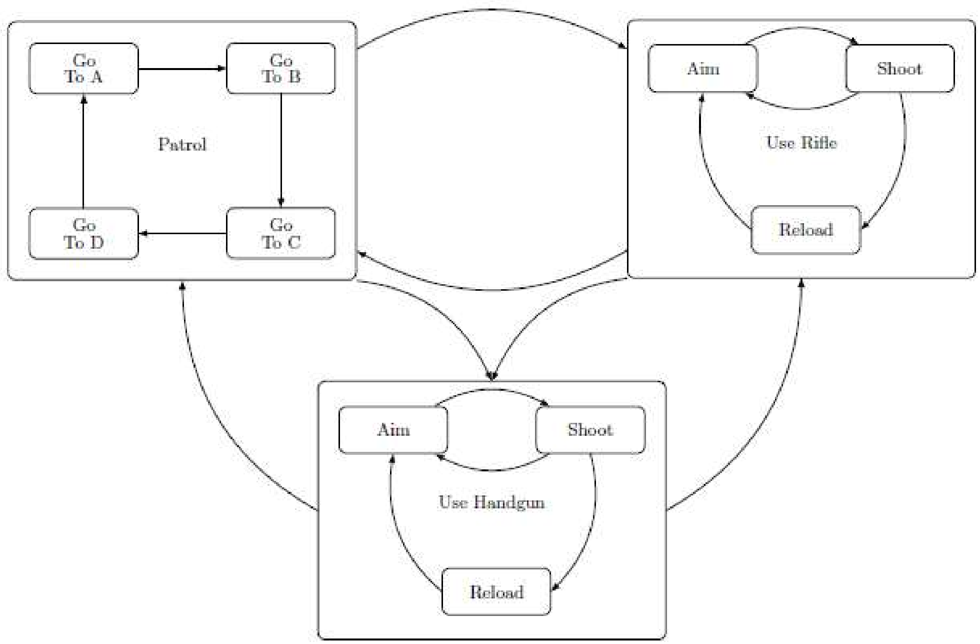
– **Maintainability**: Adding or removing states requires the re-evaluation a potentially large number of transitions and internal states of the FSM

– **Scalability**: FSMs with many states and many transitions between them are hard to modify, for both humans and computers.

– **Reusability**: The transitions between states may depend on internal variables, making it unpractical to reuse the same sub-FSM in multiple projects

M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

BTs vs HFSMs

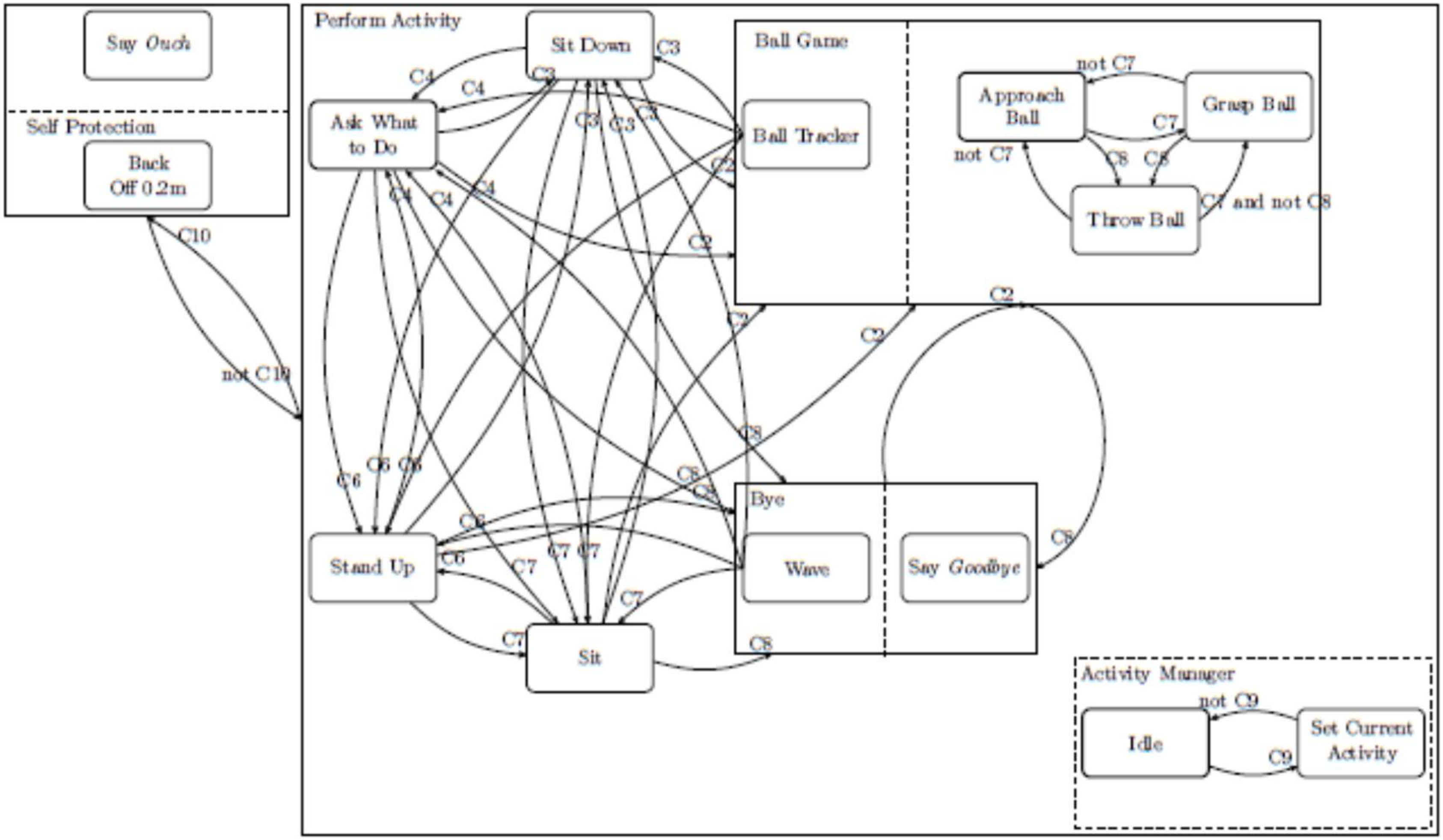


– **Maintainability**: Adding or removing states is complex. Long sequence of actions and interactions requires a fully connected subgraph

- **Manually created hierarchy**: The hierarchy resolves some problems, but a reactive HFSM results in some sub graphs being fully connected with many possible transitions

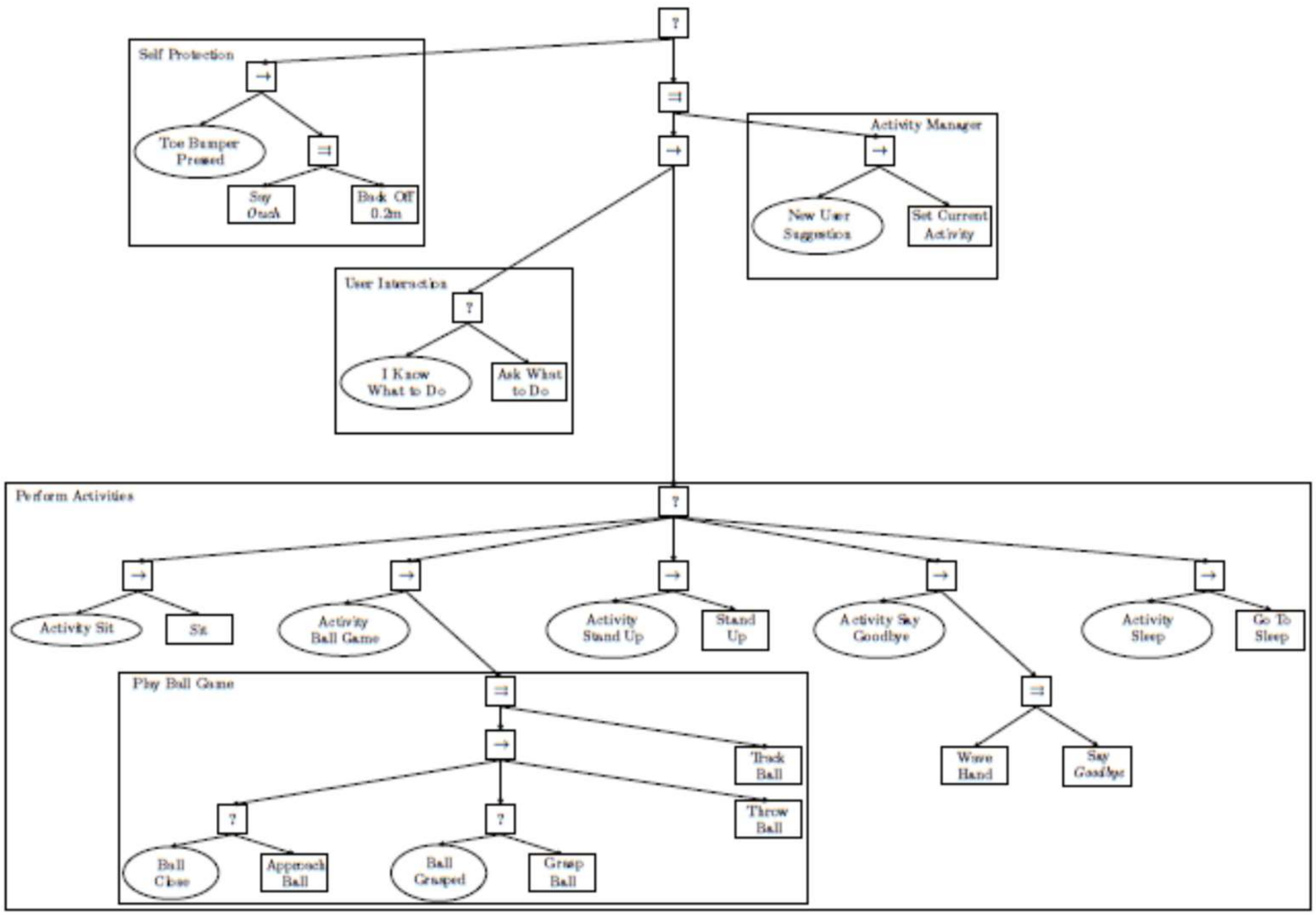
M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

BTs vs HFSMs



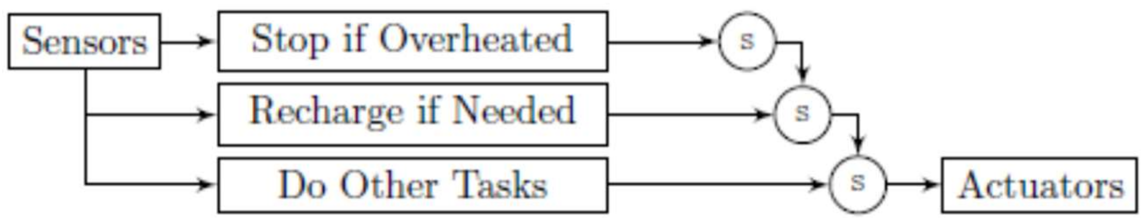
M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

BTs vs HFSMs



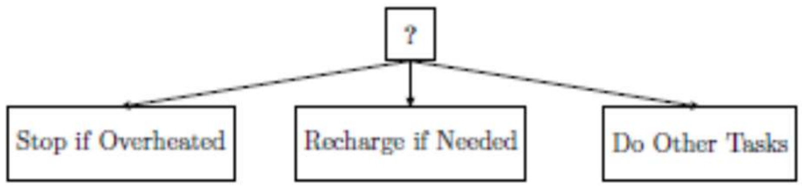
M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

BTs vs Subsumption



Fallback composition can be used to obtain an equivalent BT:

- Given a Subsumption architecture, an equivalent BT can be obtained by arranging the controllers as actions under a Fallback composition, in order from higher to lower priority
- the return status of the actions be Failure (if they do not need to execute) or Running. They never return Success



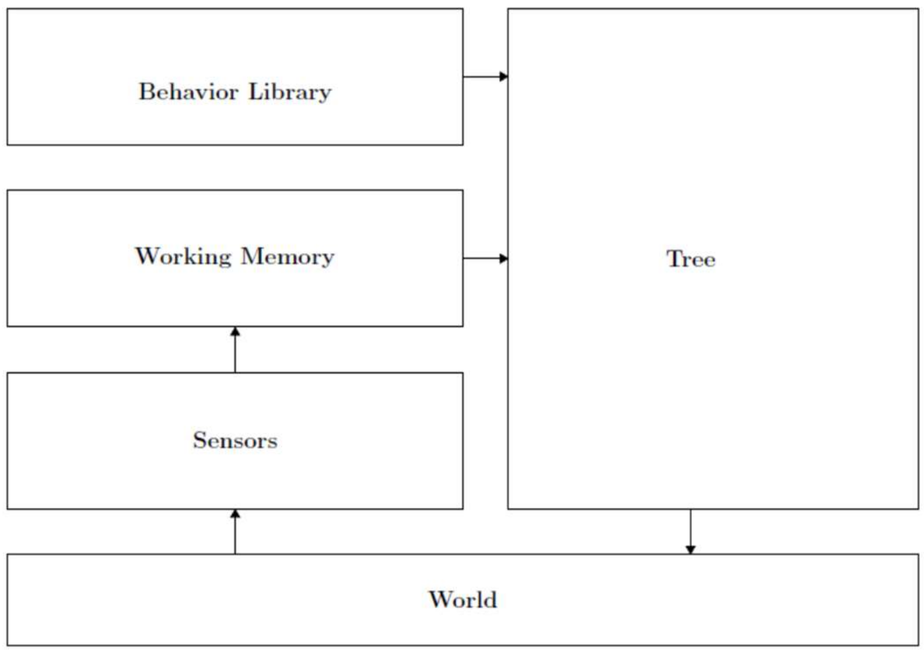
M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

ABL

Planning with a Behavior Language (ABL) [BG Weber et al 2011] for the façade video game:

- Players interact with the couple by wandering through their apartment and interacting with objects or by chatting with them directly

ABL Agent:



Michael Mateas and Andrew Stern. A Behavior Language for Story-based Believable Agents

ABL

Planning with a Behavior Language (ABL) [BG Weber et al 2011] for the façade video game:

- Players interact with the couple by wandering through their apartment and interacting with objects or by chatting with them directly

ABL Agent:

```
sequential behavior OpenDoor() {
  precondition {
    (KnockWME doorID :: door)
    (PosWME spriteID == door pos :: doorPos)
    (PosWME spriteID == me  pos :: myPos)
    (Util.computeDistance(doorPos, myPos) > 100)
  }
  specificity 2;
  // Too far to walk, yell for knocker to come in
  subgoal YellAndWaitForGuestToEnter(doorID);
}

sequential behavior OpenDoor() {
  precondition { (KnockWME doorID :: door) }
  specificity 1;
  // Default behavior - walk to door and open
  . . .
}
```

```
parallel behavior
YellAndWaitForGuestToEnter(int doorID) {
  precondition { (CurrentTimeWME t :: startT) }
  context_condition {
    (CurrentTimeWME t <= startT + 10000) }
  number_needed_for_success 1;

  with success_test {
    (DoorOpenWME door == doorID) } wait;
  with (persistent) subgoal YellForGuest(doorID);
}
```

Michael Mateas and Andrew Stern. A Behavior Language for Story-based Believable Agents

ABL

Planning with a Behavior Language (ABL) [BG Weber et al 2011]

ABL Agent:

Algorithm 8: main loop - input(initial ABL tree)

```
𝒯 ← ParallelNode
for subgoal in initial − tree do
    𝒯_g ← GetBT(subgoal)
    𝒯.AddChild(𝒯_g)
while True do
    Execute(𝒯)
```

Michael Mateas and Andrew Stern. A Behavior Language for Story-based Believable Agents

ABL

Planning with a Behavior Language (ABL) [BG Weber et al 2011]

ABL Agent:

Algorithm 9: GetBT - input(goal)

$\mathcal{T}_g \leftarrow \emptyset$
if *goal.behavior is sequential* **then**
    $\mathcal{T}_g \leftarrow$ SequenceNode
else
    $\mathcal{T}_g \leftarrow$ ParallelNode
Instructions ←`GetInstructions`(*goal*)
for *instruction in Instructions* **do**
    switch *instruction* **do**
        case *act* **do**
            $\mathcal{T}_g$.`AddChild` (ActionNode(act))
        case *mental act* **do**
            $\mathcal{T}_g$.`AddChild` (ActionNode(mental act))
        case *spawngoal* **do**
            $\mathcal{T}_g$.`AddChild` (PlaceholderNode(spawngoal))
if *goal.precondition is not empty* **then**
    $\mathcal{T}_{g'} \leftarrow$ SequenceNode
    for *proposition in precondition* **do**
        $\mathcal{T}_{g'}$.`AddChild`(ConditionNode(*proposition*))
    $\mathcal{T}_{g'}$.`AddChild`($\mathcal{T}_g$)
    return $\mathcal{T}_{g'}$
else
    return $\mathcal{T}_g$

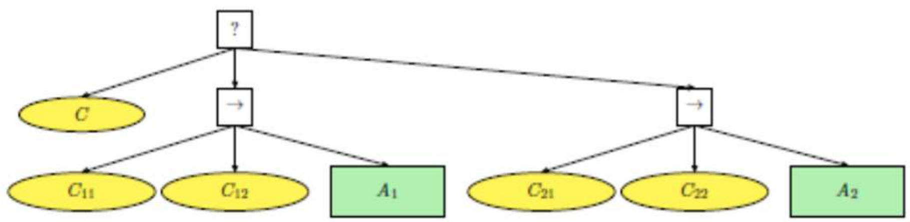
Michael Mateas and Andrew Stern. A Behavior Language for Story-based Believable Agents

PA-BT

Planning and Acting using Behavior Trees (PA-BT)

Actions	Preconditions	Postconditions
A_1	$C_{11}^{Pre}, C_{12}^{Pre}, \ldots$	$C_{11}^{Post}, C_{12}^{Post}, \ldots$
A_2	$C_{21}^{Pre}, C_{22}^{Pre}, \ldots$	$C_{21}^{Post}, C_{22}^{Post}, \ldots$
$\vdots$	$\vdots$	$\vdots$

$C_1^{Goal}, C_2^{Goal}, \ldots$



M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

PA-BT

Planning and Acting using Behavior Trees (PA-BT)

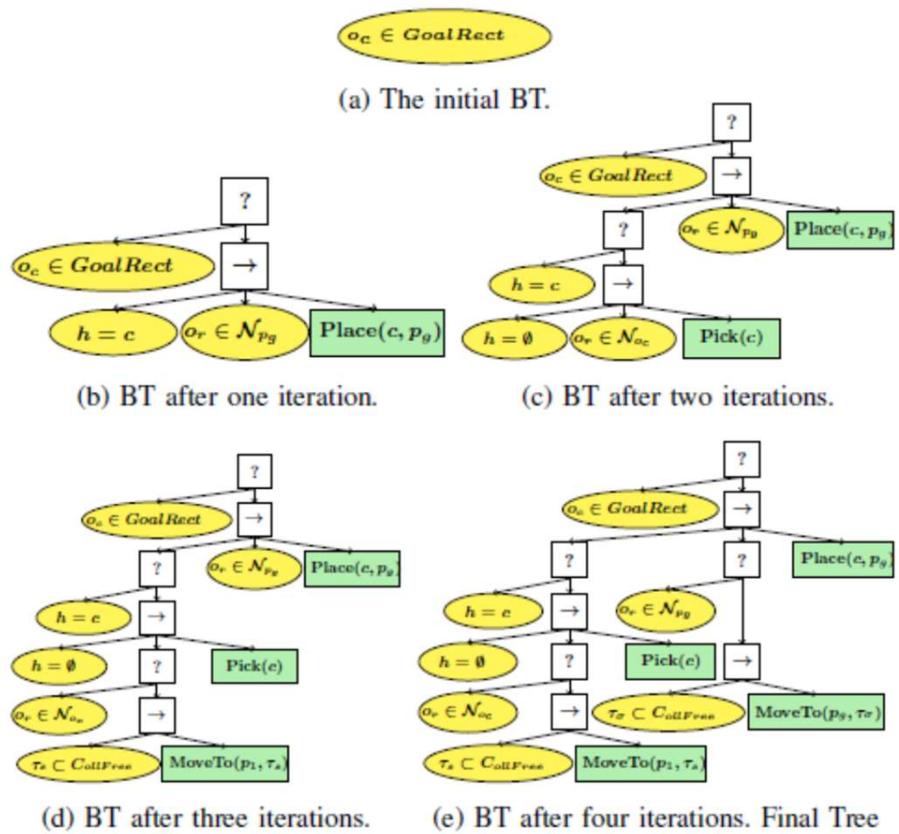


Fig. 6. BT updates during the execution.

M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

PA-BT

Planning and Acting using Behavior Trees (PA-BT)

Algorithm 1: Main Loop, finding conditions to expand and resolve conflicts

```
T ← ∅
for c in C_goal do
    T ← SequenceNode(T, c)
while True do
    T ← RefineActions(T)
    do
        r ← Tick(T)
    while r ≠ Failure
    c_f ← GetConditionToExpand(T)
    T, T_new_subtree ← ExpandTree(T, c_f)
    while Conflict(T) do
        T ← IncreasePriority(T_new_subtree)
```

Algorithm 2: Replace failed condition with new Atomic BT

```
Function ExpandTree(T, c_f)
    A_T ← GetAllActTemplatesFor(c_f)
    T_fall ← c_f
    for a in A_T do
        T_seq ← ∅
        for c_a in a.con do
            T_seq ← SequenceNode(T_seq, c_a)
        T_seq ← SequenceNode(T_seq, a)
        T_fall ← FallbackNode(T_fall, T_seq)
    T ← Substitute(T, c_f, T_fall)
    return T, T_fall
```

Back chaining: starting from a goal condition select actions to achieve that goal

M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

PA-BT

Planning and Acting using Behavior Trees (PA-BT)

Algorithm 1: Main Loop, finding conditions to expand and resolve conflicts

$\mathcal{T} \leftarrow \emptyset$
for c *in* $\mathcal{C}_{goal}$ **do**
    $\mathcal{T} \leftarrow$SequenceNode($\mathcal{T}$, c)
while *True* **do**
    $\mathcal{T} \leftarrow$`RefineActions(`$\mathcal{T}$`)`
    do
        $r \leftarrow$ `Tick(`$\mathcal{T}$`)`
    while $r \neq$ Failure
    $c_f \leftarrow$ `GetConditionToExpand(`$\mathcal{T}$`)`
    $\mathcal{T}, \mathcal{T}_{new\_subtree} \leftarrow$ `ExpandTree(`$\mathcal{T}, c_f$`)`
    while *Conflict*($\mathcal{T}$) **do**
        $\mathcal{T} \leftarrow$ `IncreasePriority(`$\mathcal{T}_{new\_subtree}$`)`

Algorithm 3: Get Condition to Expand

Function `GetConditionToExpand(`$\mathcal{T}$`)`
    for c_{next} *in* `GetConditionsBFS()` **do**
        if c_{next}*.status* = Failure *and* $c_{next} \notin$ *ExpandedNodes* **then**
            ExpandedNodes.`push_back(`c_{next}`)`
            return c_{next}
    return $None$

Back chaining: starting from a goal condition select actions to achieve that goal

M. Colledanchise and P. Ogren Behavior Trees in Robotics and AI. An Introduction

Behavior Networks

Hierarchical Behavior-based systems [Nicolescu and Mataric 2002]

Abstract and Primitive Behaviors

- **world preconditions** - conditions that activate the behaviors based on a particular state of the environment.

- **sequential preconditions** - task-dependent conditions that must be met before activating the behavior:
  - *Permanent preconditions*: preconditions that must be met during the entire execution of the behavior
  - *Enabling preconditions*: preconditions that must be met immediately before the activation of a behavior
  - *Ordering constraints*: preconditions that must have been met at some point before the behavior is activated

- **Sequential execution:** for the task segments containing temporal ordering constraints;
- **Opportunistic execution:** for the task segments for which the order of execution does not matter.

Behavior Networks

Hierarchical Behavior-based systems [Nicolescu and Mataric 2002]

Abstract and Primitive Behaviors

NETWORK-DESCRIPTION =
$<$ **Number of components (N),**
$\{$**Component-Description**$\}_N$,
Topology-Description $>$

where,

Component-Description =
$<$ **AB-Description | ABN-Description** $>$

AB-Description =
$<$ **Component-ID,**
BehaviorID, Number of Parameters (P),
$\{$**Parameter Name, Parameter Value**$\}_P$ $>$

NAB-Description =
$<$ **Component-ID,**
NETWORK-DESCRIPTION $>$

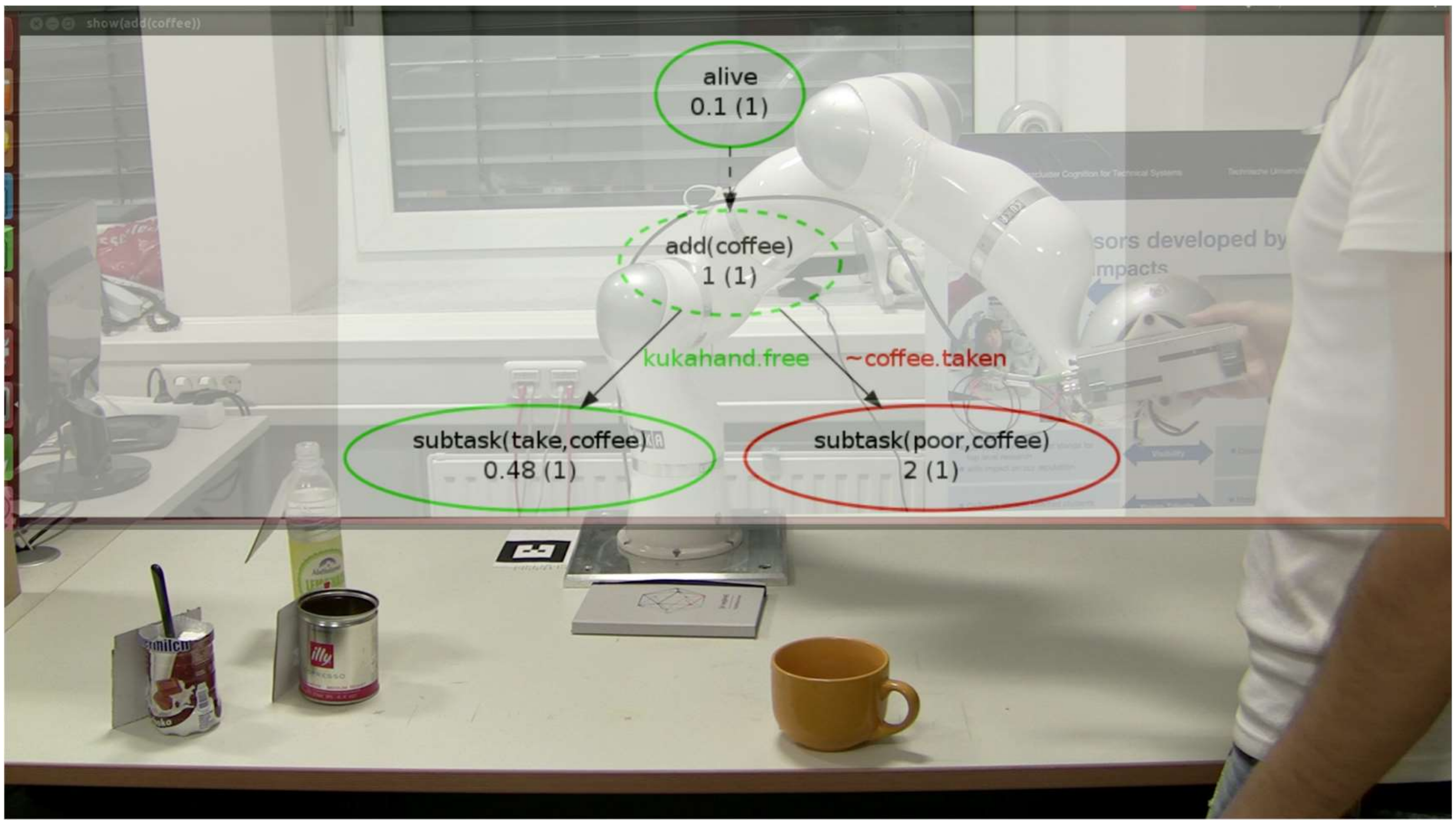
Topology-Description =
$<$ **Number of Links (L),**
$\{$**FromComp-ID, ToComp-ID, Link-Type**$\}_L$ $>$

Link-Type =
$<$ **Ordering | Enabling | Permanent** $>$

Behavior Hierarchy

Hierarchical Behavior-based systems [Caccavale Finzi 2015, 2017, 2019]

Abstract and Primitive Behaviors (and activations)



BDI Systems

[Bratman,1987]. Intention, Plans, and Practical Reason.

- BDI model inspired by the Michael Bratman's theory of human practical reasoning:
  - resource-bounded agent
  - intention and desire are proactive, intentions as commitments

Core concepts

Beliefs = information the agent has about the world
Desires = state of affairs that the agent would wish to bring about
Intentions = desires (or actions) that the agent has committed to achieve

Belief: the agent knowledge about about the world (belief set)
Desires: motivational state, objectives, tasks to be acheived (goals)
Intentions: desires with commitment, i.e. plans ready for the execution (plans)

BDI Systems

BDI particularly compelling because:

- **philosophical component** - based on a theory of rational actions in humans
- **software architecture** - it has been implemented and successfully used in a number of complex fielded applications
  - IRMA - Intelligent Resource-bounded Machine Architecture
  - PRS - Procedural Reasoning System
- **logical component** - the model has been rigorously formalized in a family of BDI logics
  - Rao & Georgeff, Wooldrige
  - $(\text{Int}\ A_i\ \varphi) \rightarrow \neg(\text{Bel}\ A_i\ \varphi)$

Practical Reasoning Agents: Deliberation: Intentions and Desires

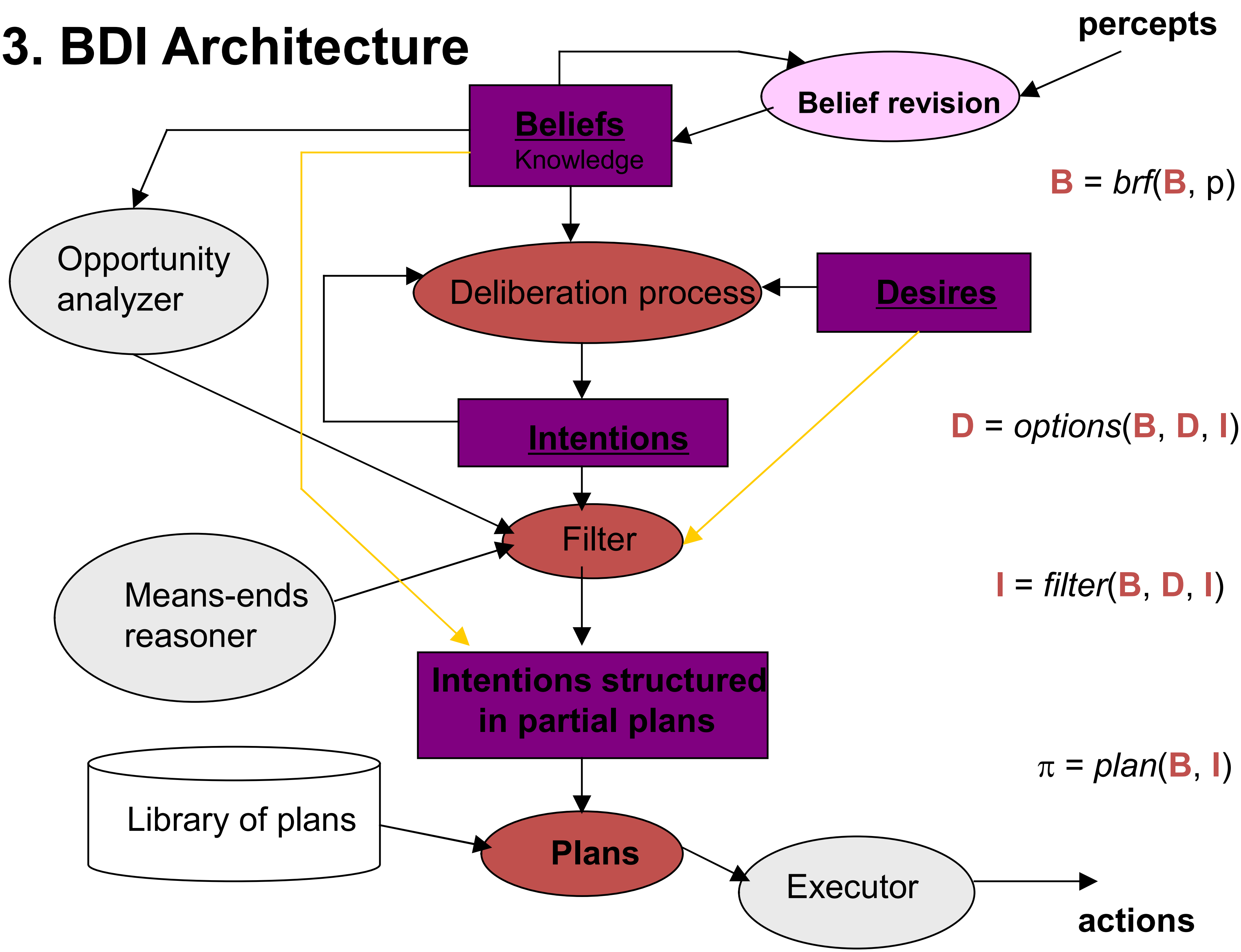
- intentions are stronger than desires

  - "My desire to play basketball this afternoon is merely a potential influencer of my conduct this afternoon. It must vie with my other relevant desires [. . .] before it is settled what I will do. In contrast, once I intend to play basketball this afternoon, the matter is settled: I normally need not continue to weigh the pros and cons. When the afternoon arrives, I will normally just proceed to execute my intentions." *[Bratman, 1990]*

Practical Reasoning Agents: <u>Intentions</u>

1. agents are expected to **<u>determine ways of achieving</u>** intentions
   - *If I have an intention to Φ, you would expect me to devote resources to deciding how to bring about Φ*
2. agents **<u>cannot</u>** adopt intentions which **<u>conflict</u>**
   - *If I have an intention to Φ , you would not expect me to adopt an intention Ψ that was incompatible with Φ*
3. agents are inclined to **<u>try again</u>** if their attempts to achieve their intention fail
   - *If an agent's first attempt to achieve Φ fails, then all other things being equal, it will try an alternative plan to achieve Φ*
4. agents **<u>believe</u>** their intentions are **<u>possible</u>**
   - *That is, they believe there is at least some way that the intentions could be brought about.*
5. agents do **<u>not believe</u>** they will **<u>not bring about</u>** their intentions
   - *It would not be rational of me to adopt an intention to Φ if I believed that I would fail with Φ*
6. under certain circumstances, agents **<u>believe</u>** they **<u>will bring about</u>** their intentions
   - *If I intend Φ, then I believe that under "normal circumstances" I will succeed withΦ*
7. agents need **<u>not intend</u>** all the expected **<u>side effects</u>** of their intentions
   - *I may believe that going to the dentist involves pain, and I may also intend to go to the dentist — but this does not imply that I intend to suffer pain!*

3. BDI Architecture



Practical Reasoning Agents

- agent control loop

```
while true

    observe the world;

    update internal world model;

    deliberate about what intention to achieve next;

    use means-ends reasoning to get a plan for the
  intention;

    execute the plan

end while
```

Practical Reasoning Agents

- agent control loop

```
while true
    observe the world;
    update internal world model;
    deliberate about what intention to achieve next;
    use means-ends reasoning to get a plan for the
  intention;
    execute the plan
end while
```

- what are the **options** (**desires**) ?
- how to **choose** an option ?
- incl. **filter**
- chosen option ➔ **intention** …

Practical Reasoning Agents

- agent control loop

```
while true

    observe the world;

    update internal world model;

    deliberate about what intention to achieve next;

    use means-ends reasoning to get a plan for the
  intention;

    execute the plan

end while
```

- what are the **options** (**desires**) ?
- how to **choose** an option ?
- incl. **filter**
- chosen option ➔ **intention** …

- when to reconsider intentions !?

Implementing Practical Reasoning Agents

- Let’s make the algorithm more formal:

Agent Control Loop Version 2

$B := B_0$; /* initial beliefs */
while true do
    get next percept ρ;
    $B := brf(B, \rho)$;
    $I := deliberate(B)$;
    $\pi := plan(B, I)$;
    $execute(\pi)$
end while

Implementing Practical Reasoning Agents

- Optimal behaviour if
  - deliberation and means-ends reasoning take a small amount of time;
  - the world is guaranteed to remain static while the agent is deliberating and performing means-ends reasoning;
  - an intention that is optimal when achieved at time t_0 (the time at which the world is observed) is guaranteed to remain optimal until time t_2 (the time at which the agent has found a course of action to achieve the intention).

Deliberation

- The *deliberate* function can be decomposed into two distinct functional components:
  - *option generation*
    the agent generates a set of possible alternatives. A function, *options*, takes the agent's current beliefs and current intentions, and from them determines a set of options (= *desires*)
  - *filtering*
    the agent chooses between competing alternatives, and commits to achieving them. In order to select between competing options, an agent uses a *filter* function.

Deliberation

Agent Control Loop Version 3

$B := B_0$;

$I := I_0$;

while true do

  get next percept ρ;

  $B := brf(B, \rho)$;

  $D := options(B, I)$;

  $I := filter(B, D, I)$;

  $\pi := plan(B, I)$;

  $execute(\pi)$

end while

Practical Reasoning Agents

If an option has successfully passed trough the filter function and is chosen by the agent as an intention, we say that **the agent has made a commitment to that option.**

Commitment implies temporal persistence of intentions; once an intention is adopted, it should not be immediately dropped out.

How committed an agent should be to its intentions?

- degrees of commitments
  - blind commitment
    - ≈ fanatical commitment: continue until achieved
  - single-minded commitment
    - continue until achieved or no longer possible
  - open-minded commitment
    - continue until no longer believed possible

Commitment Strategies

- An agent has commitment both
  - to *ends* (i.e., the wishes to bring about)
  - and *means* (i.e., the mechanism via which the agent wishes to achieve the state of affairs)

- current version of agent control loop is overcommitted, both to means and ends

  ➔ modification: *replan* if ever a <u>plan</u> goes wrong

```
Agent Control Loop Version 4
    B := B0;
    I := I0;
    while true do
        get next percept ρ;
        B := brf(B, ρ);
        D := options(B, I);
        I := filter(B, D, I);
        π := plan(B, I);
        while not empty(π) do
            α := hd(π);
            execute(α);
            π := tail(π);
            get next percept ρ;
            B := brf(B, ρ);
            if not sound(π, I, B) then
                π := plan(B, I)
            end-if
        end-while
    end-while
```

← *Reactivity, replan*

“Blind commitment”

Commitment Strategies

- this version still overcommitted to intentions:
  - never stops to consider whether or not its intentions are appropriate

➔ modification: stop for determining whether intentions have succeeded or whether they are impossible:

"*Single-minded commitment*"

Single-minded Commitment

```
Agent Control Loop Version 5
B := B_0;
I := I_0;
while true do
    get next percept ρ;
    B := brf(B, ρ);
    D := options(B, I);
    I := filter(B, D, I);
    π := plan(B, I);
    while not empty(π)
            or succeeded(I, B)
            or impossible(I, B)) do
        α := hd(π);
        execute(α);
        π := tail(π);
        get next percept ρ;
        B := brf(B, ρ);
        if not sound(π, I, B) then
            π := plan(B, I)
        end-if
    end-while
end-while
```

Dropping intentions that are impossible or have succeeded

Reactivity, replan

Intention Reconsideration

- Our agent gets to reconsider its intentions when:
  - it has <u>completely executed a plan</u> to achieve its current intentions; or
  - it believes it has <u>achieved its current intentions</u>; or
  - it believes its <u>current intentions are no longer possible</u>.

→ This is limited in the way that it permits an agent to *reconsider* its intentions

➔ modification:

Reconsider intentions after executing every action

"*Open-minded commitment*"

Open-minded Commitment

```
Agent Control Loop Version 6
1.
2.  B := B_0;
3.  I := I_0;
4.  while true do
5.      get next percept ρ;
6.      B := brf(B, ρ);
7.      D := options(B, I);
8.      I := filter(B, D, I);
9.      π := plan(B, I);
10.     while not (empty(π)
                or succeeded(I, B)
                or impossible(I, B)) do
11.         α := hd(π);
12.         execute(α);
13.         π := tail(π);
14.         get next percept ρ;
15.         B := brf(B, ρ);
16.         D := options(B, I);
17.         I := filter(B, D, I);
18.         if not sound(π, I, B) then
19.             π := plan(B, I)
20.         end-if
21.     end-while
22. end-while
```

Intention Reconsideration

- But intention reconsideration is *costly*!
  A dilemma:
  - an agent that does not stop to reconsider its intentions sufficiently often will continue attempting to achieve its intentions even after it is clear that they cannot be achieved, or that there is no longer any reason for achieving them
  - an agent that *constantly* reconsiders its attentions may spend insufficient time actually working to achieve them, and hence runs the risk of never actually achieving them
- Solution: incorporate an explicit ***meta-level control*** component, that decides whether or not to reconsider

Agent Control Loop Version 7

```
    B := B0;
    I := I0;
    while true do
        get next percept ρ;
        B := brf(B, ρ);
        D := options(B, I);
        I := filter(B, D, I);
        π := plan(B, I);
        while not (empty(π)
                or succeeded(I, B)
                or impossible(I, B)) do
            α := hd(π);
            execute(α);
            π := tail(π);
            get next percept ρ;
            B := brf(B, ρ);
            if reconsider(I, B) then
                D := options(B, I);
                I := filter(B, D, I);
            end-if
            if not sound(π, I, B) then
                π := plan(B, I)
            end-if
        end-while
end-while
```

meta-level control

Possible Interactions

- The possible interactions between meta-level control and deliberation are:

Situation number	Chose to deliberate?	Changed intentions?	Would have changed intentions?	$reconsider(\ldots)$ optimal?
1	No	—	No	Yes
2	No	—	Yes	No
3	Yes	No	—	No
4	Yes	Yes	—	Yes

Intention Reconsideration

- Situations
  - In situation (1), the agent did not choose to deliberate, and as consequence, did not choose to change intentions.
    Moreover, if it *had* chosen to deliberate, it would not have changed intentions.
    the *reconsider(…)* function is behaving optimally.
  - In situation (2), the agent did not choose to deliberate, but if it had done so, it *would* have changed intentions.
    the *reconsider(…)* function is not behaving optimally.
  - In situation (3), the agent chose to deliberate, but did not change intentions.
    the *reconsider(…)* function is not behaving optimally.
  - In situation (4), the agent chose to deliberate, and did change intentions.
    the *reconsider(…)* function is behaving optimally.
- An important assumption: cost of *reconsider(…)* is *much* less than the cost of the deliberation process itself.

Optimal Intention Reconsideration

- Kinny and Georgeff's experimentally investigated effectiveness of intention reconsideration strategies
- Two different types of reconsideration strategy were used:
  - *bold* agents
    never pause to reconsider intentions, and
  - *cautious* agents
    stop to reconsider after every action
- *Dynamism* in the environment is represented by the *rate of world change*, γ

Optimal Intention Reconsideration

- Results (not surprising):
  - If γ is low (i.e., the environment does not change quickly), bold agents do well compared to cautious ones.
    - cautious ones waste time reconsidering their commitments while bold agents are busy working towards — and achieving — their intentions.
  - If γ is high (i.e., the environment changes frequently), cautious agents tend to outperform bold agents.
    - they are able to recognize when intentions are doomed, and also to take advantage of serendipitous situations and new opportunities when they arise.

Implemented BDI Agents: IRMA

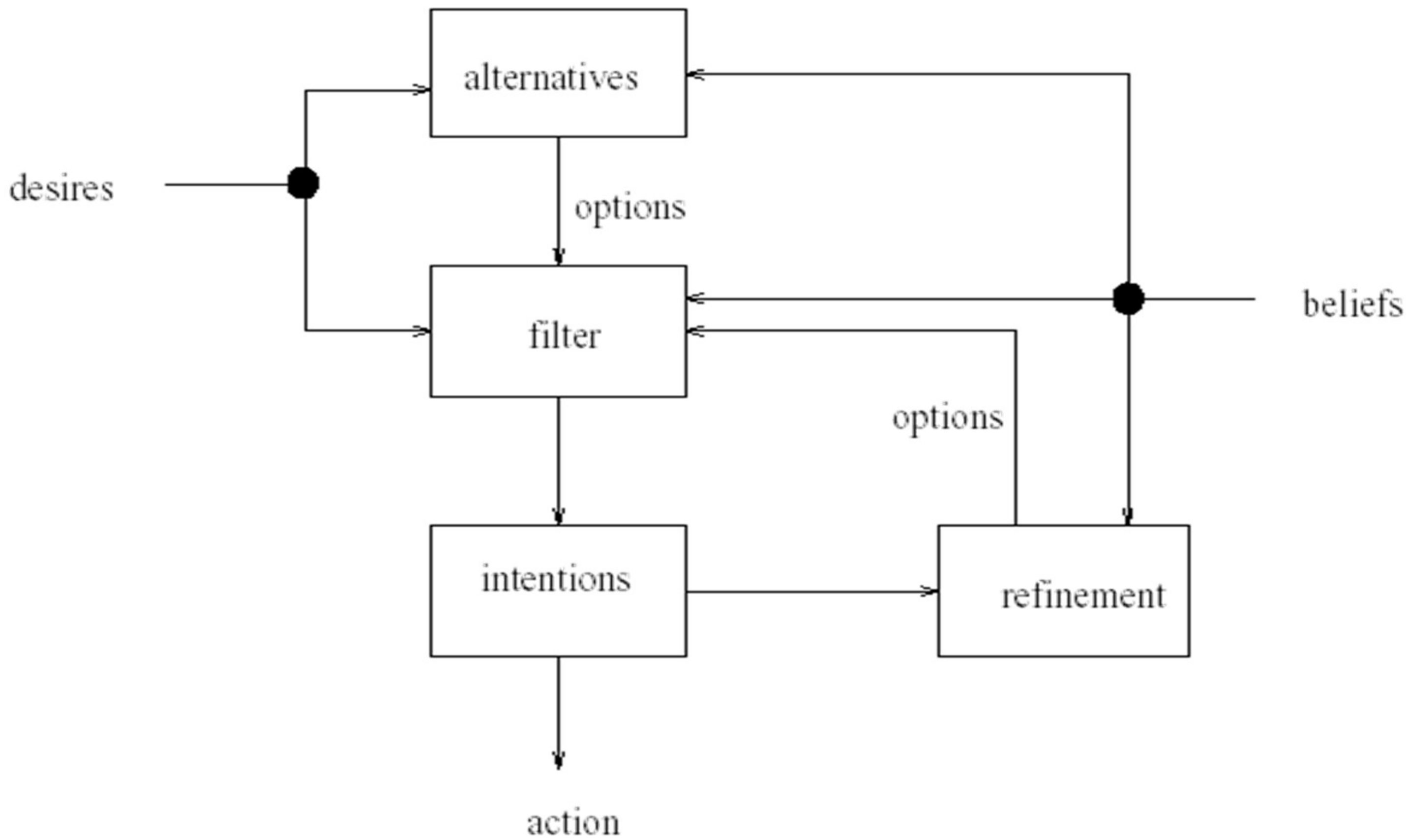
- IRMA – Intelligent Resource-bounded Machine Architecture – Bratman, Israel, Pollack

- IRMA has four key symbolic data structures:
  - a *plan library*
  - explicit representations of
    - *beliefs*: information available to the agent — may be represented symbolically, but may be simple variables
    - *desires*: those things the agent would *like* to make true — think of desires as *tasks* that the agent has been allocated;
    - *intentions*: desires that the agent has *chosen* and *committed to*

IRMA

- Additionally, the architecture has:
  - a *reasoner*
    - for reasoning about the world; an inference engine
  - a *means-ends analyzer*
    - determines which plans might be used to achieve intentions
  - an *opportunity analyzer*
    - monitors the environment, and as a result of changes, generates new options
  - a *filtering process*
    - determines which options are compatible with current intentions
  - a *deliberation process*
    - responsible for deciding upon the 'best' intentions to adopt

IRMA

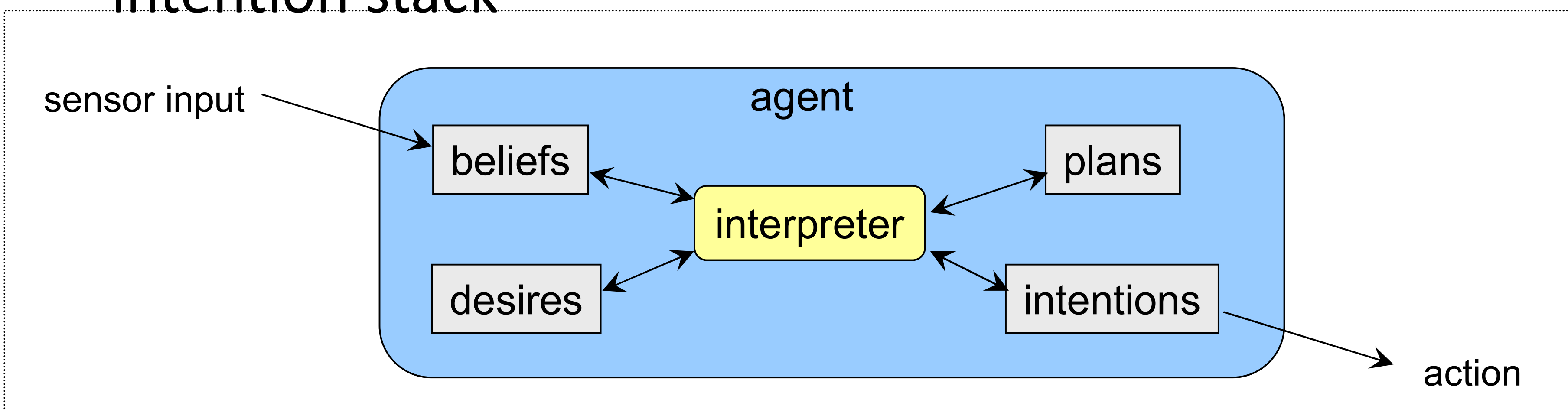

alternatives
desires
options
filter
beliefs
options
intentions
refinement
action

Practical Reasoning Agents: Procedural Reasoning System (PRS)

– "BDI-architecture" (*beliefs / desires / intentions*)
  – explicit data structures for b/d/i

– planning
  – no "on-the-fly" planning → plan libraries
    » a plan: `goal` (post-condition)
      `context` (pre-condition)
      `body` (sequence of actions / subgoals)

– intention stack



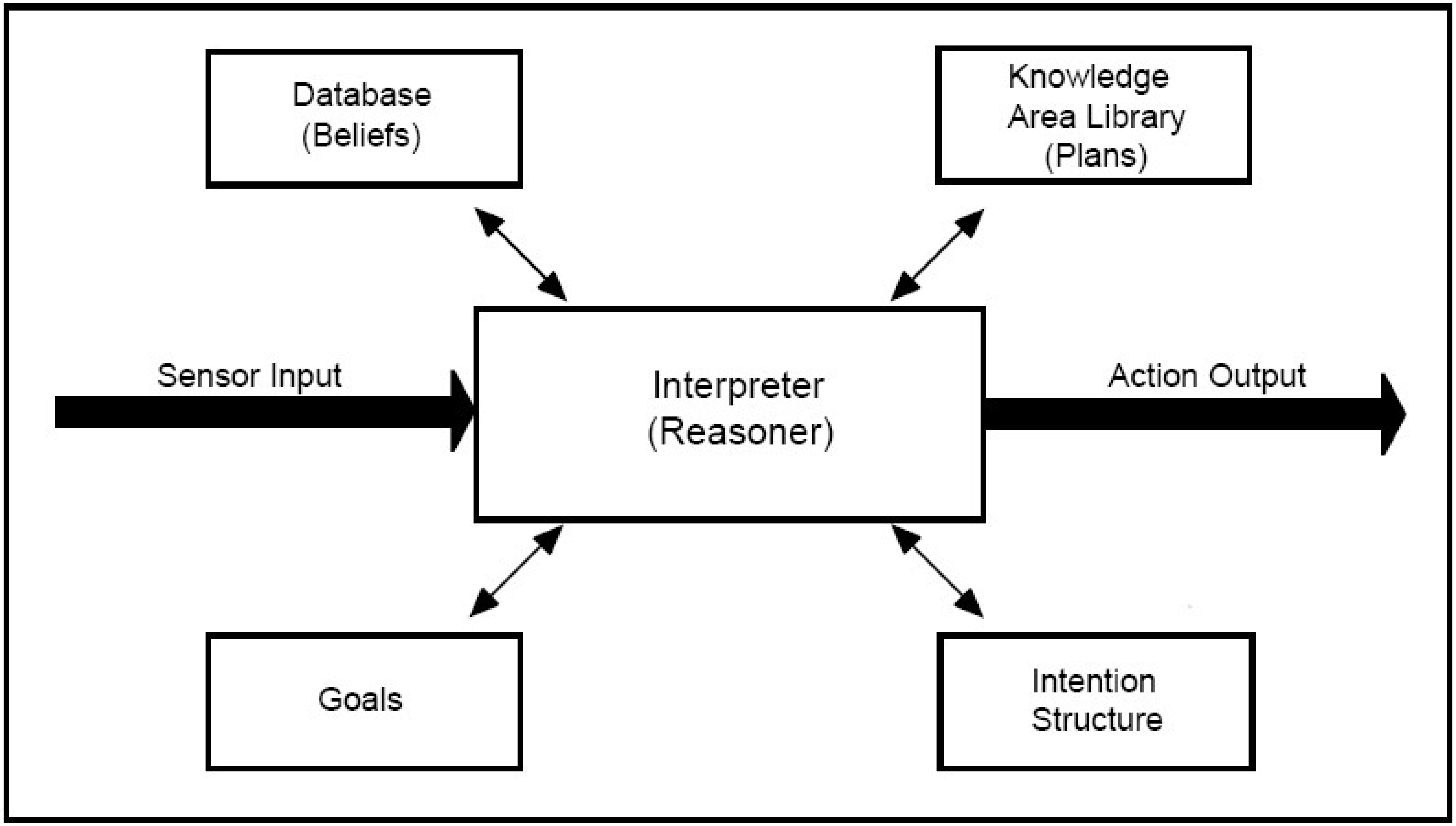
Procedural Reasoning System (PRS)

- Framework for symbolic reactive control systems in dynamic environments
  - Eg. Mobile robot control
  - Eg. Diagnosis of the Space Shuttle's Reaction Controls System

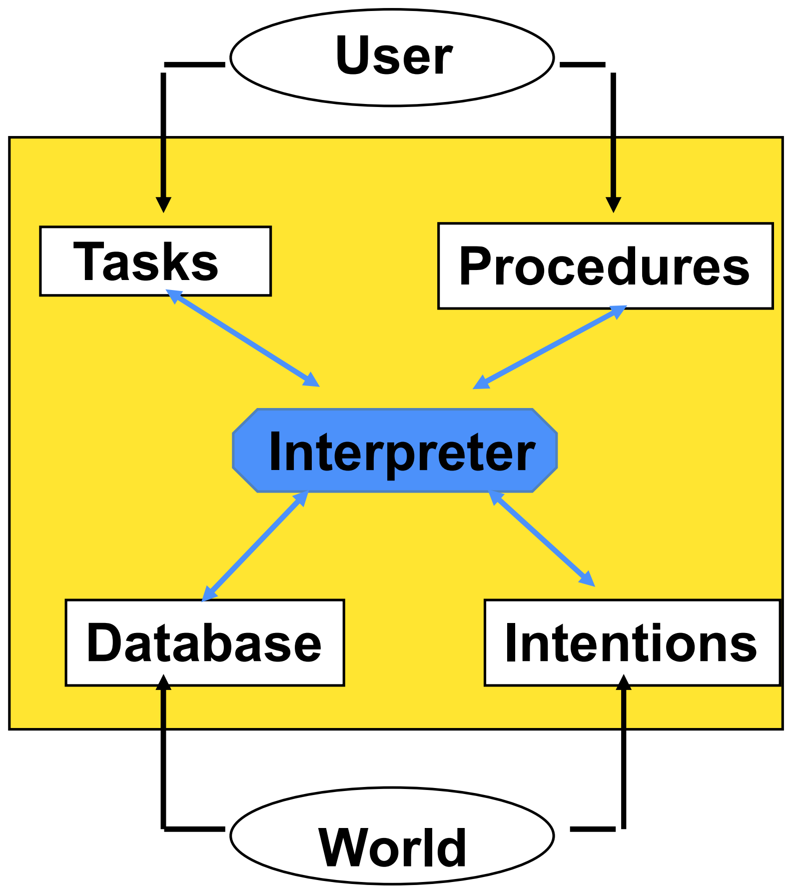
PRS: Main Features

- Pre-compiled procedural knowledge
- BDI (Belief, Desires, Intentions) foundation
- Combines deliberative and reactive features
  - Plan selection, formation, execution, sensing
- Plans dynamically and incrementally
- Integrates goal-directed and event-driven behavior
- Can interrupt plan execution
- Meta-level reasoning
- Multi-agent planning

PRS Architecture

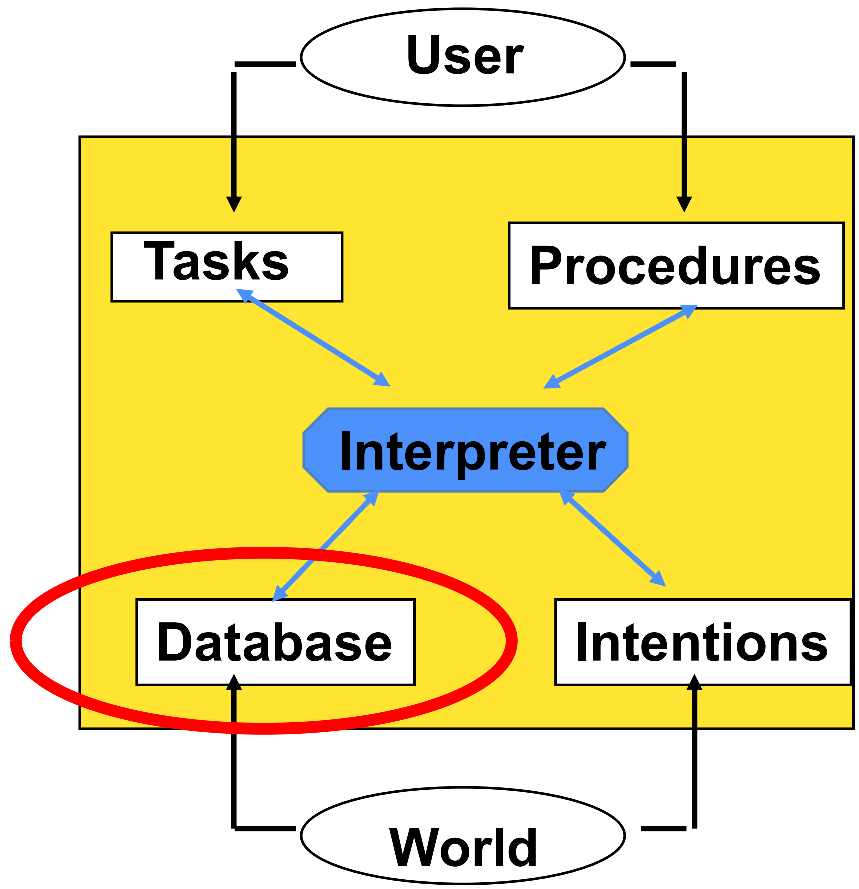


PRS Architecture



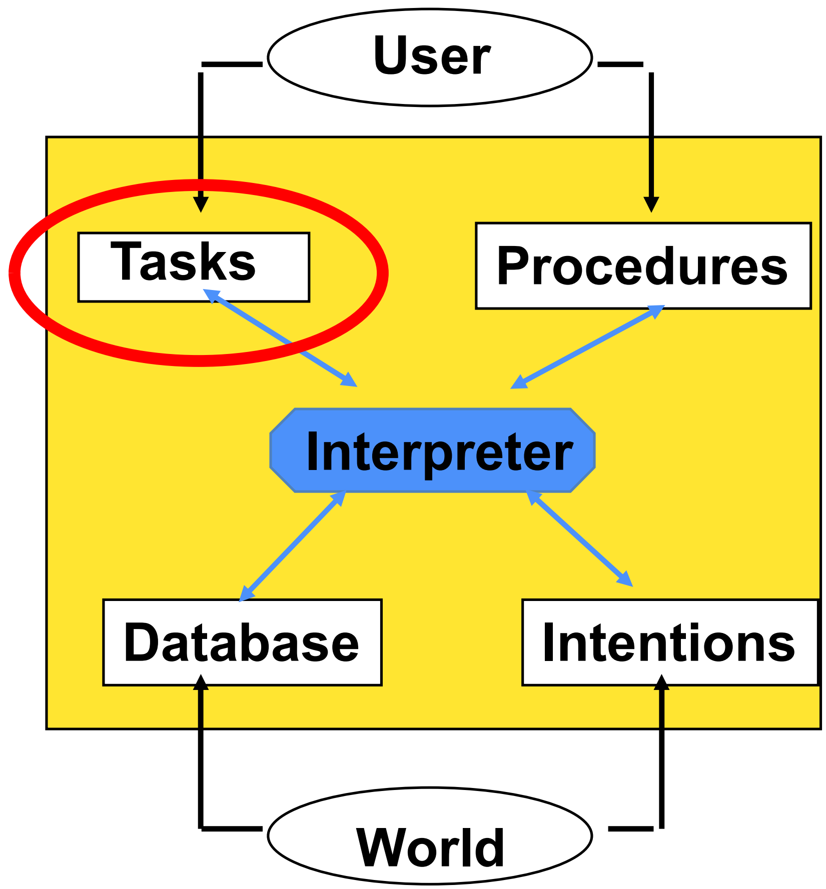
PRS Architecture: Database

- Contains beliefs or facts about the world
- Includes meta-level information
  – Eg goal G is active



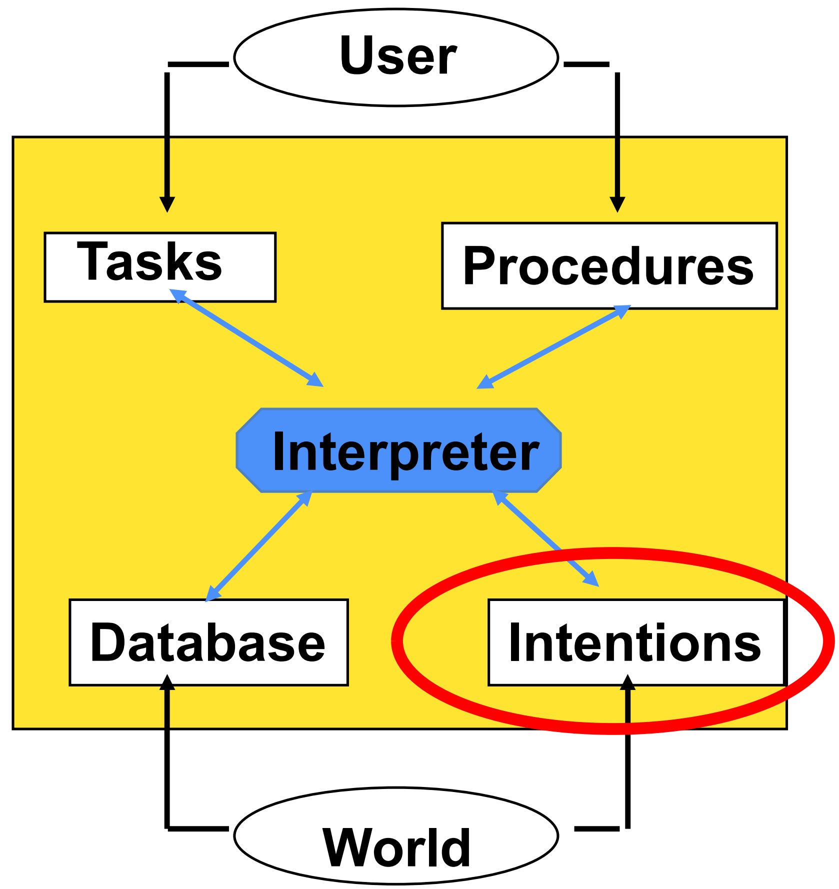
PRS Architecture: Tasks

- Represent desired behavior
- Conditions over some time interval
  - eg (walk a b): set of behaviors in which agent walks from a to b)



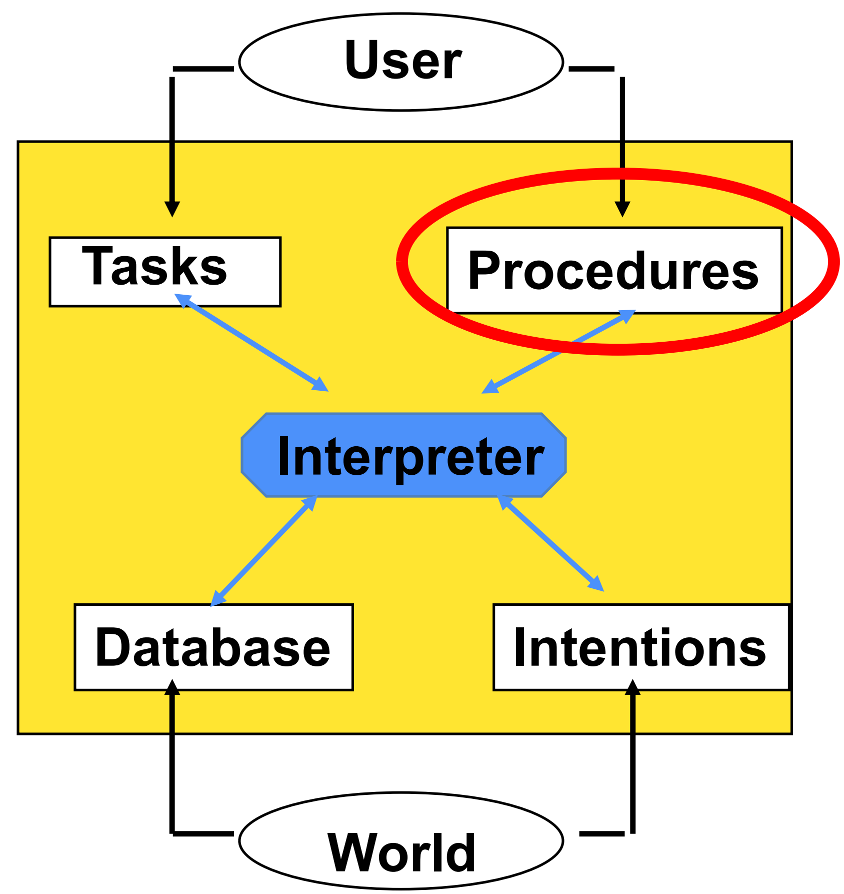
PRS Architecture: Intentions

- Currently active procedures
- Procedure currently being executed



PRS Architecture: Procedures

- Pre-compiled procedures
- Express actions and tests to achieve goals or to react to conditions



PRS

- Beliefs, goals, intentions and plan library
- Agent perceive the world through external events
- Plans – proceedural knowledge
  - Recipes for action (tree labeled with actions and formulas which evaluate to a boolean)
  - Trigger (what an agent must perceive)
  - Context (what an agent must believe)
  - If plans cannot proceed they post internal events

From Plans to Intentions

- Agents respond to Internal and External events by selecting an appropriate plan in its plan based whose trigger and context is true
- When a plan is adopted it becomes an intention
- This intention become part of the agent's intention structure

PRS Operation

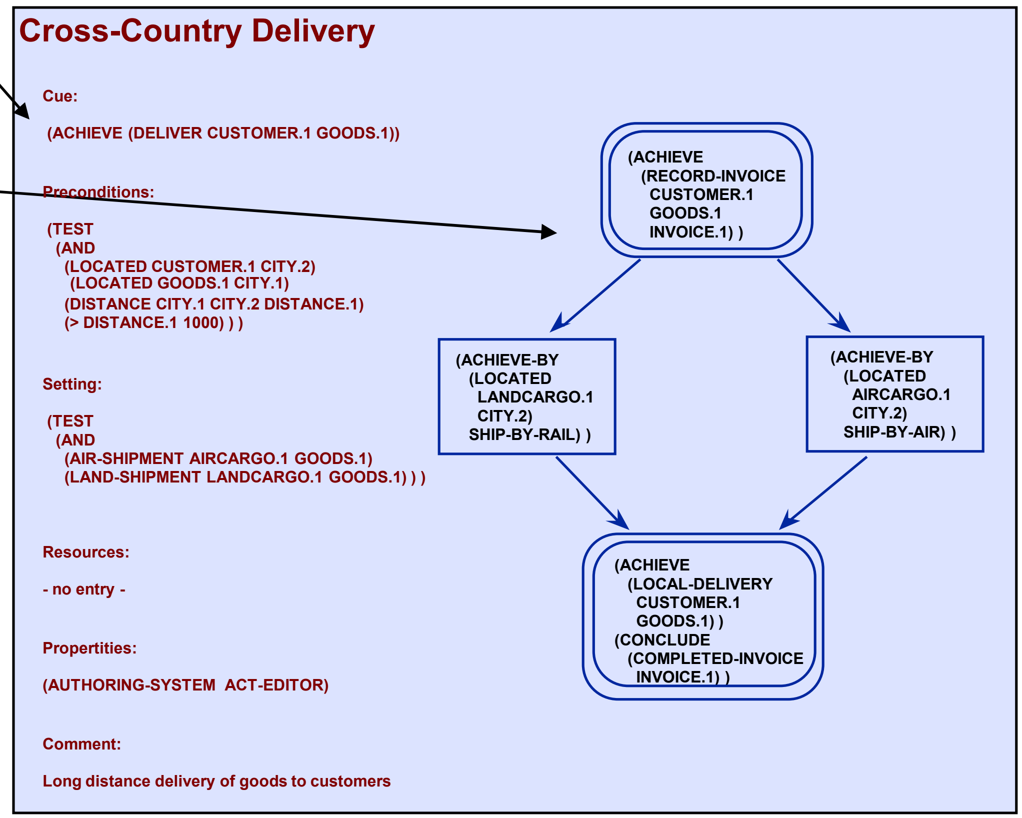
1. Perceive the world, and update the set of events.
2. For each event, generate the set of plans whose trigger condition matches the event. These are known as the *relevant* plans of an event.
3. For each event, select the subset of relevant plans whose context condition is satisfied by the agent's current beliefs. These plans are known as *active* plans.
4. From the set of active plans, select one for execution so that it is now an intention.
5. Include this new intention in the current intention structure either by creating a new intention stack or by placing it on the top of an existing stack.
6. Select an intention stack, take the topmost intention, and execute the next formula in it.

Expressing Tasks in a Dynamic Environment

- (! P) -- achieve P
- (? P) -- test P
- (# P) -- maintain P
- (^ C) -- wait until C
- (-> C) -- assert C
- (~> C) -- retract C

Representing Procedures with Act Formalism

- **Environment conditions**
  - Purpose (goal or condition)
  - applicability criteria
- **Plot**
  - directed graph
  - partially ordered conditional & parallel actions, loops
  - Successful node execution by achievement of node's goals
  - If no body: primitive action



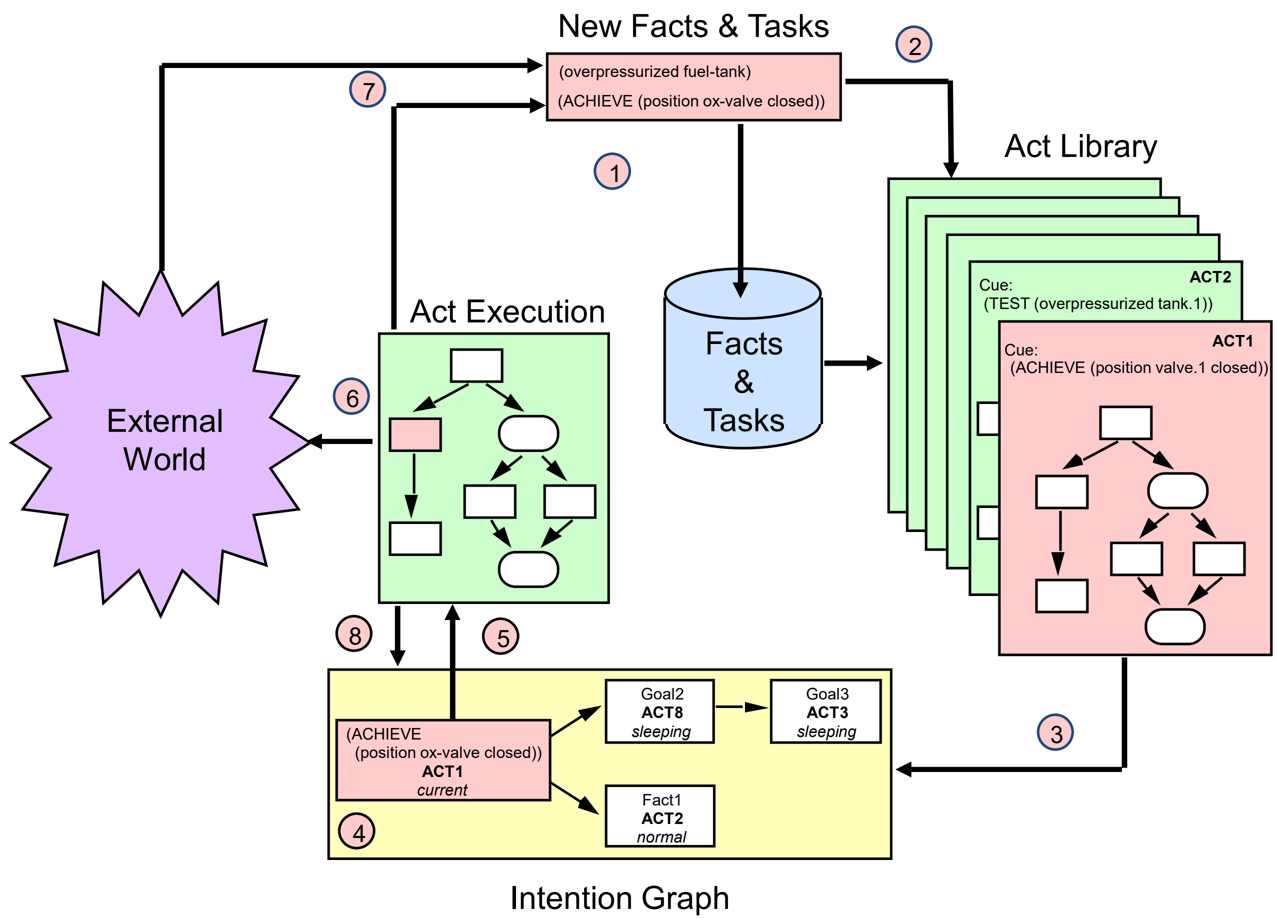
Metapredicates

- Achieve
- Test
- Wait-Until
- Require-Until
- Achieve-By {proc}
- Conclude {effects}
- Use-Resource

PRS Interpreter

Execution Cycle

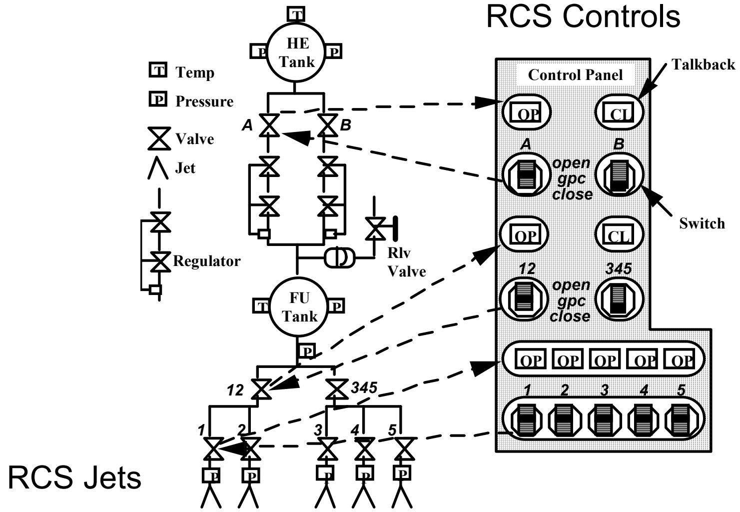
1. New information arrives that updates facts and/or tasks
2. Acts are triggered by new facts or tasks
3. A triggered Act is intended
4. An intended Act is selected
5. That intention is activated
6. An action is performed
7. New facts or tasks are posted
8. Intentions are updated

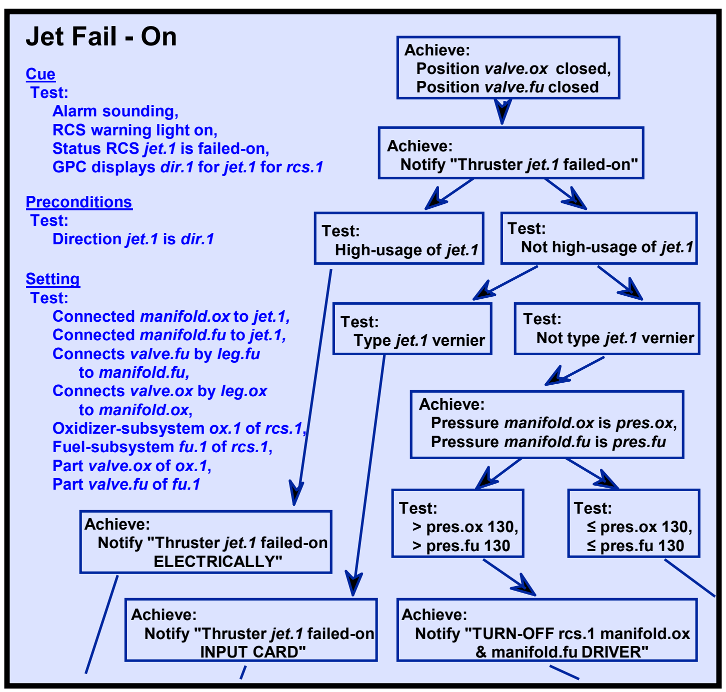


Meta-Reasoning

- Can include meta-level procedures
  - eg: choose among multiple applicable procedures
  - eg: evaluate how much more reasoning can be done within time constraints
  - eg: how to achieve a conjunction or disjunction of goals

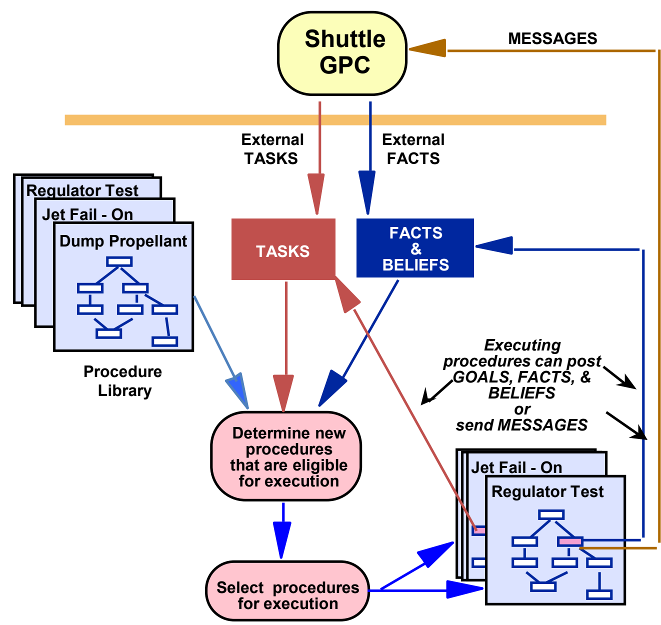
Shuttle's RCS Malfunction Handling





PRS's procedure library

- Automates specification and execution of RCS (Reaction Control System) malfunction procedures.
- Reacts to changes in RCS. Ensures safe operation while carrying out diagnosis and remediation procedures.



PRS

Versions of PRS:

- UM-PRS
- OpenPRS (formerly C-PRS and Propice)
- AgentSpeak
- Distributed Multi-Agent Reasoning System (dMARS)
- JAM
- JACK Intelligent Agents
- SRI Procedural Agent Realization Kit (SPARK)
- PRS-CL

Multiple Tasks, Multiple Agents

- Multithreaded operation: multiple tasks being performed, runtime stacks where tasks are executed, suspended, and resumed
- Supports distributed planning: several PRS agents run asynchronously and communicate through message passing

Model-based Programming

- High level programming vs. Planning
- Model-based execution and diagnosis

- RMPL [Kim et. al 2001, Williams et al. 2003]
- GOLOG [Hahnel et al. 1998, Finzi Orlandini 2005, Carbone et al. 2008]

Summary

- Reactive Action Packages (RAPs)
  - Networks of “conditions” and “tasks”
- ESL (Execution Support Language)
- Task Control Architecture (TCA)
  - Network arranged according to “vertical capabilities”
- Claraty and PLEXIL
- Behavior Trees
- BDI
  - IRMA
  - Procedural Reasoning System (PRS)