Progression of Basic Action Theories in the Situation Calculus:
4 Results and an Application in Video Games

Stavros Vassos

Department of Computer Science
University of Toronto

Joint work with Hector Levesque and Gerhard Lakemeyer

September 23, 2008
Overview

- Introduction
- Progression in the situation calculus
- Proving a 10 year-old conjecture: a negative result
- Implications
- Three positive results
- Conclusions

- A (possible) application in video games
- Practical AI modules for video games
The problem we examine lies in the field of *reasoning about action and change*. Given a logical formalism that is able to:

1. represent the current state of the world;
2. represent the dynamics of the world;
3. answer queries about the current state and the possible future states of the world.

*Problem:* How to update the representation of the current state after action execution.
Think of a non-player character (NPC) in video game equipped with such a logical formalism.

1 Representing the current state:
   - “the door is locked”
   - “the red and green levers are up and the yellow lever is down”

2 Representing the dynamics of the world:
   - “pushing/pulling a lever toggles its up/down position”
   - “pressing the button unlocks the door, provided that all levers are in up position”

3 Answering queries about the future:
   - “is there a sequence of actions such that after executing it the door will be unlocked?”
Basic action theories in the situation calculus

The *situation calculus* is a first-order logical language with limited second-order features:

- $S_0$ is the initial situation;
- $do(a, S_0)$ is the resulting situation after action $a$ has been performed in $S_0$;
- *fluents* are like normal predicates but also depend on a situation argument: $locked(S_0)$, $\neg leverUp(yellow, do(a, S_0))$. 

A basic action theory (BAT) consists essentially of:

- $\text{KB}$: a first-order knowledge base (possibly with quantifiers) that represents what holds in the initial situation $S_0$;
- $\text{DYN}$: a set of first-order action precondition and effect axioms that represent the dynamics of the world.
Basic action theories in the situation calculus

The *situation calculus* is a first-order logical language with limited second-order features:

- $S_0$ is the initial situation;
- $do(a, S_0)$ is the resulting situation after action $a$ has been performed in $S_0$;
- *fluents* are like normal predicates but also depend on a situation argument: $locked(S_0)$, $\neg leverUp(yellow, do(a, S_0))$.

A *basic action theory (BAT)* consists essentially of:

- **KB**: a first-order knowledge base (possibly with quantifiers) that represents what holds in the initial situation $S_0$;
- **DYN**: a set of first-order action precondition and effect axioms that represent the dynamics of the world.
The NPC is equipped with a BAT that represents the previous scenario.

Queries about the future are formed as entailment questions:

- $BAT \models locked(do(a_1, S_0))$ \hspace{1cm} $[a_1]$
- $BAT \models \neg locked(do(a_2, do(a_1, S_0)))$ \hspace{1cm} $[a_1, a_2]$
The NPC is equipped with a BAT that represents the previous scenario.

Queries about the future are formed as entailment questions:

- $\text{BAT} \models locked(\text{do}(a_1, S_0))$ \quad [a_1]
- $\text{BAT} \models \neg locked(\text{do}(a_2, do(a_1, S_0)))$ \quad [a_1, a_2]

The NPC chooses to *execute* actions $[a_1, a_2]$ that result in unlocking the door.

We want a *new BAT* that *progresses* $S_0$ to the current state.

This is necessary for *long-lived* agents, like our NPC.
Problem: progression of basic action theories

$BAT : \text{KB (knowledge base about } S_0) + \text{DYN (axioms for dynamics)}, \text{ action } a \text{ is executed by the robot.}$

$BAT' : \text{new KB'} + \text{same DYN},$

such that $BAT'$ entails the same first-order sentences about the future (after $a$ has been executed) as $BAT$. 
Problem: progression of basic action theories

\( \textit{BAT} : \text{KB (knowledge base about } S_0 \text{)} + \text{DYN (axioms for dynamics), action } a \text{ is executed by the robot.} \)

\( \textit{BAT'}: \text{new KB'} + \text{same DYN, such that } \textit{BAT'} \text{ entails the same first-order sentences about the future (after } a \text{ has been executed) as } \textit{BAT}. \)

What is a \textit{correct} definition for KB'?
Problem: progression of basic action theories

\( BAT \): KB (knowledge base about \( S_0 \)) + DYN (axioms for dynamics), action \( a \) is executed by the robot.

\( BAT' \): new KB’ + same DYN,
such that \( BAT' \) entails the same first-order sentences about the future (after \( a \) has been executed) as \( BAT \).

What is a correct definition for KB’?

- LR-progression [Lin & Reiter 1997]:
  - model-theoretic specification of KB’;
  - positive result: it is correct;
  - negative result: there are theories for which there is no first-order representation of KB’.
Problem: progression of basic action theories

**BAT**: KB (knowledge base about \( S_0 \)) + DYN (axioms for dynamics), action \( a \) is executed by the robot.

**BAT'**: *new* KB' + *same* DYN, such that BAT' entails the same *first-order* sentences about the future (after \( a \) has been executed) as BAT.

What is a *correct* definition for KB'?

- **LR-progression** [Lin & Reiter 1997]:
  - model-theoretic specification of KB';
  - positive result: it is correct;
  - negative result: there are theories for which there is *no first-order* representation of KB'.

- **FO-progression** [Pednault 1987]:
  - the specification of KB' is based on *first-order* entailments;
  - [Lin & Reiter 1997] conjecture: FO-progression is too weak;
  - Result 1: proof of the conjecture!
Problem: progression of basic action theories

\( BAT \): KB (knowledge base about \( S_0 \)) + DYN (axioms for dynamics), action \( a \) is executed by the robot.

\( BAT' \): new KB' + same DYN, such that \( BAT' \) entails the same first-order sentences about the future (after \( a \) has been executed) as \( BAT \).

What is a correct definition for KB'?

- LR-progression [Lin & Reiter 1997]:
  
  second-order but always correct

- FO-progression [Pednault 1987]:
  
  first-order but Lin & Reiter conjectured it is incorrect
Result 1: *FO*-progression is weak in the general case

<table>
<thead>
<tr>
<th>$BAT$</th>
<th>facts about $S_0$</th>
<th>facts about $S_1$</th>
<th>facts about $S_2$, ⋯</th>
</tr>
</thead>
<tbody>
<tr>
<td>$BAT$</td>
<td>FO entailments of KB</td>
<td>FO entailments of KB+DYN</td>
<td>FO entailments of KB+DYN</td>
</tr>
<tr>
<td>$BAT'$</td>
<td>–</td>
<td>FO entailments of KB’</td>
<td>FO entailments of KB’+DYN</td>
</tr>
</tbody>
</table>
Result 1: FO-progression is weak in the general case

<table>
<thead>
<tr>
<th>BAT</th>
<th>facts about $S_0$</th>
<th>facts about $S_1$</th>
<th>facts about $S_2, \ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAT</td>
<td>FO entailments of KB</td>
<td>FO entailments of KB+DYN</td>
<td>FO entailments of KB+DYN</td>
</tr>
<tr>
<td>BAT′</td>
<td>–</td>
<td>FO entailments</td>
<td>FO entailments of KB+DYN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of KB′</td>
<td></td>
</tr>
</tbody>
</table>

We specify a BAT such that:
- KB implies a certain property about all models of BAT;
- DYN makes the property is not expressible in KB′;
- the property is expressible in a sentence $\phi$ that refers to all future situations:

So KB′+DYN $\not\models \phi$ but KB+DYN $\models \phi$, and thus FO-progression loses information.

*Theorem 1*: FO-progression is incorrect in the general case.
Result 1: *FO*-progression is weak in the general case

Some details: theory of arithmetic, non-standard models, numbers not reachable from zero, unnamed objects, …

\[ \forall x (x \neq 0 \equiv \exists y \; n(y) = x) \]
\[ \forall x \forall y (n(x) = n(y) \supset x = y) \]

**KB:**
\[ F(0, S_0) \land \forall x (F(x, S_0) \supset F(n(x), S_0)) \]
\[ \exists x \neg F(x, S_0) \]

**DYN:**
\[ F(x, do(a, s)) \equiv a = A \land x = 0 \lor a = B \land \neg F(x, s) \land \exists y (x = n(y) \land F(y, s)) \]

**KB’:**
\[ \forall x (F(x, do(A, S_0)) \equiv x = 0) \]

**ϕ:**
\[ \exists x \forall s (do(A, S_0) \sqsubseteq s \supset \neg F(x, s)) \]
Implications of Result 1

There is no general definition of progression that will work within first-order logic in all cases.

Three alternatives:

- restrict the type of axioms in KB:
  - e.g. allow only a limited form of *incomplete knowledge*;

- restrict the type of axioms in DYN:
  - e.g. allow only *STRIPS actions*;

- weaken the form of progression and restrict the queries:
  - consider a progression that is *sound but not complete* in general and specify a *class of queries* for which it is also complete;
  - e.g. consider *FO*-progression and restrict queries to refer to *specific situation only*. 
Result 2: Restricting the type of the queries

<table>
<thead>
<tr>
<th>BAT</th>
<th>BAT’ (progression)</th>
</tr>
</thead>
<tbody>
<tr>
<td>unrestricted KB</td>
<td>LR-progression KB’ is second-order in the general case, but ...</td>
</tr>
<tr>
<td>unrestricted DYN</td>
<td></td>
</tr>
<tr>
<td>unrestricted queries</td>
<td></td>
</tr>
<tr>
<td>unrestricted KB</td>
<td>... an FO-progression KB’ is as good assuming a practical restriction:</td>
</tr>
<tr>
<td>unrestricted DYN</td>
<td>limited quantification over situations.</td>
</tr>
<tr>
<td>restricted queries</td>
<td></td>
</tr>
</tbody>
</table>
Result 2: FO-progression is sometimes correct

\[ \text{BAT} : \text{unrestricted KB + unrestricted DYN,} \]
\[ \text{action a is executed by the agent.} \]
\[ \text{BAT'}: \text{first-order KB'} + \text{same DYN}. \]

**Theorem 2:** An FO-progression KB' is correct for a practical class of sentences that allows limited quantification over situations, e.g:

- invariants of the form \( \forall s \Phi(s) \),
  "in all future situations \( \Phi \) holds";
- sentences of the form \( \exists s \Phi(s) \),
  "there is a future situation where \( \Phi \) holds".
Result 2: FO-progression is sometimes correct

\begin{align*}
BAT & : \text{unrestricted KB + unrestricted DYN}, \\
& \text{action } a \text{ is executed by the agent.} \\
BAT' & : \text{first-order KB' + same DYN.}
\end{align*}

**Theorem 2:** An FO-progression KB' is **correct** for a practical class of sentences that allows limited quantification over situations, e.g:

- invariants of the form $\forall s \Phi(s)$, \\
  \text{“in all future situations $\Phi$ holds”};

- sentences of the form $\exists s \Phi(s)$, \\
  \text{“there is a future situation where $\Phi$ holds”}.

This result uses intuitions similar to those of [Savelli 2006] that studies regression in the situation calculus.
### Results 3 and 4: Restricting the type of DYN

<table>
<thead>
<tr>
<th>BAT</th>
<th>BAT' (progression)</th>
</tr>
</thead>
<tbody>
<tr>
<td>unrestricted KB</td>
<td>LR-progression KB' is second-order in the general case, but . . .</td>
</tr>
<tr>
<td>unrestricted DYN</td>
<td></td>
</tr>
<tr>
<td>unrestricted queries</td>
<td></td>
</tr>
<tr>
<td>unrestricted KB</td>
<td>. . . it is always first-order representable assuming a practical restriction: local-effects.</td>
</tr>
<tr>
<td><em>restricted DYN</em></td>
<td></td>
</tr>
<tr>
<td>unrestricted queries</td>
<td></td>
</tr>
</tbody>
</table>
Local-effect BATs

A BAT is *local-effect* iff actions may only affect the truth value of fluent atoms where arguments are also arguments of the action.

- Action *mess-with*(c, d) may only affect ground fluent atoms with arguments in \{c, d\}.
- Action *push*(x) that is intended to affect the truth value of *leverUp*(x, s) is local-effect.
  
  E.g., *push*(green) may only affect *leverUp*(green, s).
- Action *push-all* that is intended to affect all three levers is not local-effect.

Note that this is a restriction on DYN only.
Result 3 (local-effect): \( LR \)-progression is first-order

\[
BAT : \text{unrestricted KB} + \text{local-effect DYN},
\]
action \( a \) is executed by the agent.

\[
BAT' : \text{first-order KB'} + \text{same DYN}.
\]

**Theorem 3:** For any local-effect BAT we can always specify a first-order representation of the \( LR \)-progression KB'.

Stavros Vassos
September 23, 2008 15 / 27
Result 3 (local-effect): $LR$-progression is first-order

\[ BAT : \textit{unrestricted} \ KB + \textit{local-effect} \ DYN, \]
\[ \text{action } a \text{ is executed by the agent.} \]
\[ BAT' : \textit{first-order} \ KB' + \textit{same} \ DYN. \]

\textbf{Theorem 3:} For any local-effect BAT we can always specify a first-order representation of the $LR$-progression $KB'$. 

$KB'$ is equivalent to the (\textit{infinite}) set of first-order entailments of BAT that only talk about situation $do(a, S_0)$;
Result 3 (local-effect): $LR$-progression is first-order

$BAT: \textit{unrestricted KB} + \textit{local-effect DYN}$, action $a$ is executed by the agent.

$BAT': \textit{first-order KB'} + \textit{same DYN}$.

Theorem 3: For any local-effect BAT we can always specify a first-order representation of the $LR$-progression $KB'$.

$KB'$ is equivalent to the (infinite) set of first-order entailments of BAT that only talk about situation $do(a, S_0)$;

If we restrict the BATs a little more then we can specify a finite first-order $KB'$. 
Strictly Local-effect BATs

A BAT is *strictly local-effect* iff it is local-effect and moreover:
- KB is finite and has unique-names for objects;
- DYN is further restricted so that conditional effects cannot quantify over fluent atoms in the condition.

Example: action *pushbutton* will result in unlocking the door provided all levers are up.

Two ways to formalize the condition “all levers are up”:
- Local-effect: *pushbutton* affects *locked(s)* provided
  \[ \forall x(\text{lever}(x) \supset \text{leverUp}(x, s)) \]
- Strictly local-effect: *pushbutton* affects *locked(s)* provided
  \[ \text{leverUp}(\text{red}, s) \land \text{leverUp}(\text{green}, s) \land \text{leverUp}(\text{yellow}, s) \]
Result 4 (strictly local-effect): $LR$-progression is finite

**BAT:** finite, unique names KB + strictly local-effect DYN, action $a$ is executed by the agent.

**BAT′:** finite, first-order KB′ + same DYN.

**Theorem 4:** For any strictly local-effect BAT we can always specify a finite first-order representation of the $LR$-progression KB′.
Result 4 (strictly local-effect): \( LR \)-progression is finite

\[
\begin{align*}
\text{BAT} : & \quad \text{finite, unique names KB + strictly local-effect DYN,} \\
& \quad \text{action } a \text{ is executed by the agent.} \\
\text{BAT'} : & \quad \text{finite, first-order KB' + same DYN.}
\end{align*}
\]

**Theorem 4:** For any strictly local-effect BAT we can always specify a finite first-order representation of the \( LR \)-progression KB'.

KB' can be obtained by modifying the original KB in a systematic way.
Result 4 (strictly local-effect): LR-progression is finite

\[ \text{BAT} : \text{finite, unique names KB} + \text{strictly local-effect DYN,} \]
\[ \text{action } a \text{ is executed by the agent.} \]
\[ \text{BAT': finite, first-order KB'} + \text{same DYN}. \]

**Theorem 4:** For any strictly local-effect BAT we can always specify a finite first-order representation of the LR-progression KB'.

KB' can be obtained by modifying the original KB in a systematic way.

Let's see an example!
Result 4 (strictly local-effect): Example

- **KB:** $\forall x \ F(x, S_0)$
- **DYN:** action $a(x)$ flips the value of $F(x, s)$

Execute action $a(c)$ and progress by modifying the original KB.
Result 4 (strictly local-effect): Example

- KB: $\forall x \ F(x, S_0)$
- DYN: action $a(x)$ flips the value of $F(x, s)$

Execute action $a(c)$ and progress by modifying the original KB.

1. $a(c)$ may only affect the value of $F(c, S_0)$. Consider cases:

$$F(c, S_0) \land KB \lor \neg F(c, S_0) \land KB$$
Result 4 (strictly local-effect): Example

- **KB**: \( \forall x \ F(x, S_0) \)
- **DYN**: action \( a(x) \) flips the value of \( F(x, s) \)

Execute action \( a(c) \) and progress by modifying the original KB.

1. \( a(c) \) may only affect the value of \( F(c, S_0) \). Consider cases:
   \[
   F(c, S_0) \land KB \lor \neg F(c, S_0) \land KB
   \]

2. For each case unit-propagate “in a first-order way”:
   \[
   F(c, S_0) \land \forall x (x = c \land F(x, S_0) \lor x \neq c \land F(x, S_0)) \lor \\
   \neg F(c, S_0) \land \forall x (x = c \land F(x, S_0) \lor x \neq c \land F(x, S_0))
   \]
Result 4 (strictly local-effect): Example

- KB: $\forall x \, F(x, S_0)$
- DYN: action $a(x)$ flips the value of $F(x, s)$

Execute action $a(c)$ and progress by modifying the original KB.

1. $a(c)$ may only affect the value of $F(c, S_0)$. Consider cases:

$$F(c, S_0) \land KB \lor \neg F(c, S_0) \land KB$$

2. For each case unit-propagate “in a first-order way”:

$$F(c, S_0) \land \forall x (x = c \land true \lor x \neq c \land F(x, S_0)) \lor$$

$$\neg F(c, S_0) \land \forall x (x = c \land false \lor x \neq c \land F(x, S_0))$$
Result 4 (strictly local-effect): Example

- KB: $\forall x \ F(x, S_0)$
- DYN: action $a(x)$ flips the value of $F(x, s)$

Execute action $a(c)$ and progress by modifying the original KB.

1. $a(c)$ may only affect the value of $F(c, S_0)$. Consider cases:

$$F(c, S_0) \land KB \lor \neg F(c, S_0) \land KB$$

2. For each case unit-propagate “in a first-order way”:

$$F(c, S_0) \land \forall x (x = c \land \text{true} \lor x \neq c \land F(x, S_0)) \lor$$

$$\neg F(c, S_0) \land \forall x (x = c \land \text{false} \lor x \neq c \land F(x, S_0))$$

3. For each case progress the value:

$$\neg F(c, S_1) \land \forall x (x = c \land \text{true} \lor x \neq c \land F(x, S_1)) \lor$$

$$F(c, S_1) \land \forall x (x = c \land \text{false} \lor x \neq c \land F(x, S_1))$$
Result 4 (strictly local-effect): Example

- KB: \( \forall x \ F(x, S_0) \)
- DYN: action \( a(x) \) flips the value of \( F(x, s) \)

Execute action \( a(c) \) and progress by modifying the original KB.

1. \( a(c) \) may only affect the value of \( F(c, S_0) \). Consider cases:
   \[ F(c, S_0) \land KB \lor \neg F(c, S_0) \land KB \]

2. For each case unit-propagate “in a first-order way”:
   \[
   F(c, S_0) \land \forall x (x = c \land \text{true} \lor x \neq c \land F(x, S_0)) \lor \\
   \neg F(c, S_0) \land \forall x (x = c \land \text{false} \lor x \neq c \land F(x, S_0))
   \]

3. For each case progress the value:
   \[
   \neg F(c, S_1) \land \forall x (x \neq c \supset F(x, S_1))
   \]
Conclusions

- We investigated the problem of *progression* in the *situation calculus*.
- We proved one negative (Result 1) and three positive results (Result 2, 3, and 4):
  - [Vassos & Levesque, AAAI 2008],
  - [Vassos & Lakemeyer & Levesque, KR 2008].
- Result 1 justifies the second-order definition of progression by Lin and Reiter.
- The other three results show that under conditions first-order logic is adequate.
- Result 1 consists a proof for a problem that has been open since it was first formalized in [Lin and Reiter 1997].
Conclusions

- We investigated the problem of *progression* in the *situation calculus*.

- We proved one negative (Result 1) and three positive results (Result 2, 3, and 4):
  
  [Vassos & Levesque, AAAI 2008],
  [Vassos & Lakemeyer & Levesque, KR 2008].

- Result 1 justifies the second-order definition of progression by Lin and Reiter.

- The other three results show that under conditions first-order logic is adequate.

- Result 1 consists a proof for a problem that has been open since it was first formalized in [Lin and Reiter 1997].

What do these results mean *in practice*?
Conclusions

- Updating a first-order knowledge base is tricky!
  - incomplete information: “the key that opens this door is in one of these boxes”;
  - facts expressed using quantifiers: “there is no key that opens this door”.

- Under practical conditions it is feasible (Result 4):
  - whenever the effects of actions are bounded to affect a finite number of things, there is a simple way to update the KB.
A (possible) application in video games

Think of a NPC equipped with a reasoning module that is able to:
- represent the character's view of the current state of the game world and its dynamics as a situation calculus BAT;
- answer queries about the current state and the future states;
- update the (representation of the) current state as actions take place.

This can be the basis for:
- re-active behavior: decide how to act based on the representation of the current state;
- pro-active behavior: decide how to act based on projection, goal achievability, planning.

Intelligent NPCs in video games: the AI researcher perspective; the game developer perspective.
A (possible) application in video games

Think of a NPC equipped with a *reasoning module* that is able to:

- represent the character’s view of the current state of the game world and its dynamics as a situation calculus BAT;
- answer queries about the current state and the future states;

Intelligent NPCs in video games: the AI researcher perspective; the game developer perspective.
A (possible) application in video games

Think of a NPC equipped with a *reasoning module* that is able to:

- represent the character’s view of the current state of the game world and its dynamics as a situation calculus BAT;
- answer queries about the current state and the future states;
- update the (representation of the) current state as actions take place.

This can be the basis for:

- reactive behavior: decide how to act based on the representation of the current state;
- proactive behavior: decide how to act based on projection, goal achievability, planning.

Intelligent NPCs in video games: the AI researcher perspective; the game developer perspective.
A (possible) application in video games

Think of a NPC equipped with a reasoning module that is able to:

- represent the character’s view of the current state of the game world and its dynamics as a situation calculus BAT;
- answer queries about the current state and the future states;
- update the (representation of the) current state as actions take place.

This can be the basis for:

- re-active behavior: decide how to act based on the representation of the current state;
- pro-active behavior: decide how to act based on projection, goal achievability, planning.
A (possible) application in video games

Think of a NPC equipped with a *reasoning module* that is able to:

- represent the character’s view of the current state of the game world and its dynamics as a situation calculus BAT;
- answer queries about the current state and the future states;
- update the (representation of the) current state as actions take place.

This can be the basis for:

- **re-active** behavior: decide how to act based on the representation of the current state;
- **pro-active** behavior: decide how to act based on projection, goal achievability, planning.

Intelligent NPCs in video games:

- the AI researcher perspective;
- the game developer perspective.
Intelligent NPC: the AI researcher perspective

Pick a system for AI agents, implement the NPC..

..and connect it to the game world (3D game engine).
Intelligent NPC: the AI researcher perspective

Pick a system for AI agents, implement the NPC.

1. Specify the world that the NPC is situated:
   - the *changing properties* of the world and their current value;
     door1=locked, availWeapons=[w1,w2]
   - the *actions* that can take place in the world;
     kick(x), fireWeaponAt(x), move(d,s)
   - the *effects* of the actions in the world;
     kick(door) → open(door) provided ...

2. Specify the intended behavior of the robot using features such as *goals, intentions, preferences, utility functions*, as well as predefined *reactive rules, plans*, etc;

The reasoning core of the system mixes ingredients 1 and 2 appropriately to produce intelligent behavior.

..and connect it to the game world (3D game engine).
Intelligent NPC: the AI researcher perspective

reasoning module

world spec.

behavior spec.

game world
Intelligent NPC: the game developer perspective

Practical difficulties:
- it requires a lot of memory and computing time;
- the NPCs would not be responsive in real-time.
Intelligent NPC: the game developer perspective

Practical difficulties:
- it requires a lot of memory and computing time;
- the NPCs would not be responsive in real-time.

Moreover, it would be awkward to implement an NPC like this:
- typically the AI system assumes full control over the agent:
  all aspects of the NPC’s behavior need to be encoded in the AI system using the language for specifying parts 1 and 2, e.g., how about path-finding?
- in order to use the AI system for only some reasoning task you need to know the inner workings of the system.

Let’s see some more details on the implementation of an NPC in a 3D game engine.
Intelligent NPC: the game developer perspective

- 3D game engine in C++, NPC is an instance of a C++ class.
- In every frame a “thinking” function is called that specifies what the NPC should do.
Intelligent NPC: the game developer perspective

- 3D game engine in C++, NPC is an instance of a C++ class.
- In every frame a “thinking” function is called that specifies what the NPC should do.

The body of this function usually implements a finite-state-automaton as a big if-then-else block. The conditionals depend on:

- the state of the NPC, i.e. variables of the form `state=ATTACKING, energy=80;`
- sensory data, i.e. C++ functions of the form `bool enemyInSight();`
- other implemented helping functions such as `bool pathfind(from, to, options).`
Intelligent NPC: the game developer perspective

- 3D game engine in C++, NPC is an instance of a C++ class.
- In every frame a “thinking” function is called that specifies what the NPC should do.

Game developer:
```cpp
int NPC::think()
{
    if (state==IDLE &&
        energy>70 &&
        enemyInSight())
        state=ATTACKING;
    else if ...
    ...
}
```
Intelligent NPC: the game developer perspective

- 3D game engine in C++, NPC is an instance of a C++ class.
- In every frame a “thinking” function is called that specifies what the NPC should do.

Game developer:
```cpp
int NPC::think()
{
    if (state==IDLE && energy>70 && enemyInSight())
        state=ATTACKING;
    else if ...
    ...
}
```

AI researcher:
```cpp
int NPC::think()
{
    get sensory data;
    push data to the AI system;
    wait for the system to compute the next move;
    perform the next move;
}
```

**Two extremes:** either all in C++ or all in the AI system.
Practical AI modules for video games

- **Two extremes:** either all in C++ or all in the AI system.
- **Middle ground:**
  
  Identify practical *AI modules* that can replace *C++ code* that the game developer needs to implement for the behavior of the NPC.

- Historically that’s the most common way that AI techniques have been used in commercial video games.

- Most widely used AI technique:
  
  A* heuristic search, used for path-finding:
  
  - solves a particular sub-problem with clear input and output;
  - can be embedded easily to any control structure for NPCs.

- Maybe that’s the kind of AI modules we should be aiming for! (at least for applications in video games..)
Practical AI modules for video games

One last point about the use of AI techniques in video games.

- A* dates back to 1968.
- The only reasoning module ever used in a commercial game (F.E.A.R., 2005, Monolith Productions, Vivendi) is a simplified version of a STRIPS planner that dates back to 1971.
- AI techniques need feedback from the video game industry so that they evolve as needed for the video games.
Practical AI modules for video games

One last point about the use of AI techniques in video games.

- A* dates back to 1968.
- The only reasoning module ever used in a commercial game (F.E.A.R., 2005, Monolith Productions, Vivendi) is a simplified version of a STRIPS planner that dates back to 1971.
- AI techniques need feedback from the video game industry so that they evolve as needed for the video games.

Modern video games are ideal for applying AI research about agents/robots:

- detailed software worlds with advanced 3D game engines that follow realistic physics;
- many autonomous NPCs interact with the objects in the world, the player, and with each other;
- rich sensor data for each NPC.