Situation Calculus-based Online Plan Recognition in Continuous Domains

Christoph Schwering

RWTH Aachen University

December 20, 2011

Motivation

Approach

Model

proc $overtake(V, W)$
$behind(V, W)?;$
$leftLaneChange(V);$
wait for $behind(W, V);$
$rightLaneChange(V)$
$\ldots$

Interpreter

Observations

at time 0: $pos(A) = (10, -2)$
at time 1: $pos(A) = (25, -2)$
at time 2: $pos(A) = (40, 0)$
$\ldots$

Set of programs that explain the observations.

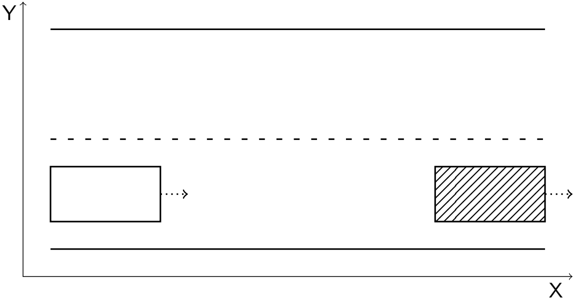
Outline

Related Work

	Kautz and Allen [1986]	Charniak and Goldman [1991]	Goultiaeva and Lespérance [2006]
Plans	consistent	likely	consistent
Tools	circumscription	Bayesian network	situation calculus
Modeling	first-order logic		ConGolog
Focus	abstraction		abstraction, online
Observations	primitive action occurrences		

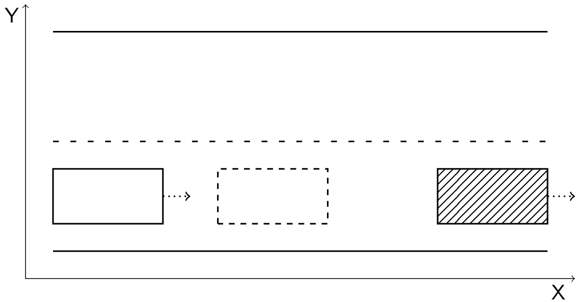
Bui et al. [2002]	Geib and Goldman [2009]	Ramirez and Geffner [2009]
likely	likely	consistent/likely
HMM	HMM	planner
hierarchical MDPs	plan tree grammars	STRIPS, goal library
abstraction, uncertainty	abstraction, partial ordering	–
primitive action occurrences		

Modeling



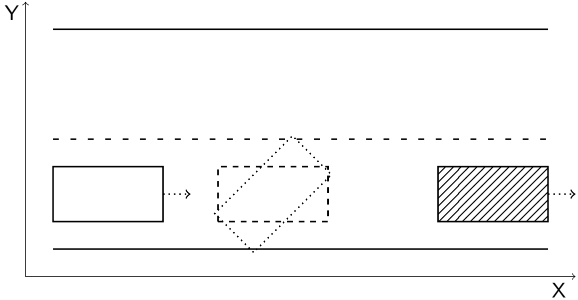
- Global cartesian view
- Vehicle = rectangle
- Instantaneous actions $setYaw$, $setVeloc$

Modeling



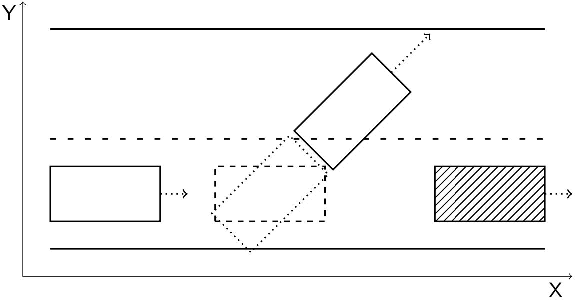
- Global cartesian view
- Vehicle = rectangle
- Instantaneous actions $setYaw$, $setVeloc$

Modeling



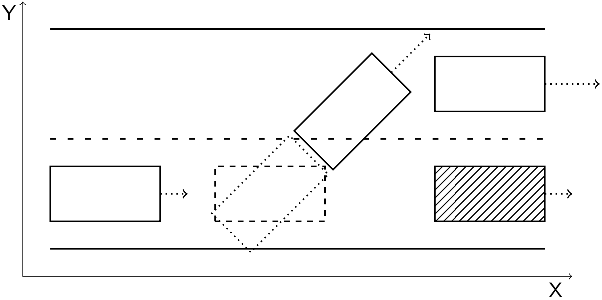
- Global cartesian view
- Vehicle = rectangle
- Instantaneous actions $setYaw$, $setVeloc$

Modeling



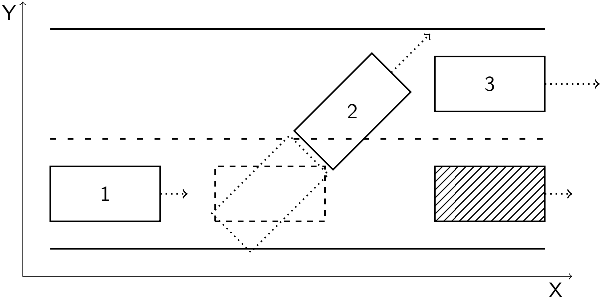
- Global cartesian view
- Vehicle = rectangle
- Instantaneous actions $setYaw$, $setVeloc$

Modeling



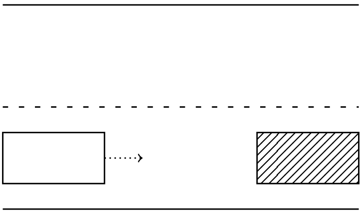
- Global cartesian view
- Vehicle = rectangle
- Instantaneous actions $setYaw$, $setVeloc$

Modeling



- Global cartesian view
- Vehicle = rectangle
- Instantaneous actions $setYaw$, $setVeloc$

Programs



proc $leftLaneChange(V)$

  pick $\gamma \in \{4°, 6°, \ldots, 12°\}$ **do**

    $setYaw(V, \gamma)$

  endpick;

  $onRightLane(V)$ **?**;

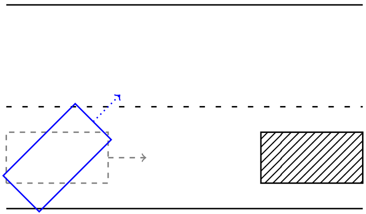
  % time passes indefinitely

  $setYaw(V, 0°)$;

  $onLeftLane(V)$ **?**

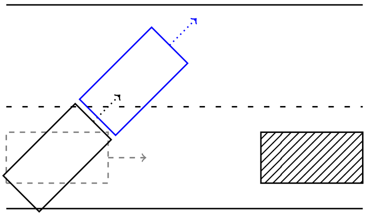
endproc

Programs



proc $leftLaneChange(V)$
  pick $\gamma \in \{4°, 6°, \ldots, 12°\}$ **do**
    $setYaw(V, \gamma)$
  endpick;
  $onRightLane(V)$ **?**;
  % time passes indefinitely
  $setYaw(V, 0°)$;
  $onLeftLane(V)$ **?**
endproc

Programs



proc $leftLaneChange(V)$

  pick $\gamma \in \{4°, 6°, \ldots, 12°\}$ **do**

    $setYaw(V, \gamma)$

  endpick;

  $onRightLane(V)$ **?**;

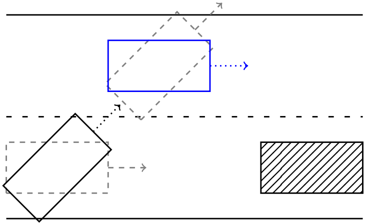
  % time passes indefinitely

  $setYaw(V, 0°)$;

  $onLeftLane(V)$ **?**

endproc

Programs



proc $leftLaneChange(V)$

  pick $\gamma \in \{4°, 6°, \dots, 12°\}$ **do**

    $setYaw(V, \gamma)$

  endpick;

  $onRightLane(V)$ **?**;

  % time passes indefinitely

  $setYaw(V, 0°)$;

  $onLeftLane(V)$ **?**

endproc

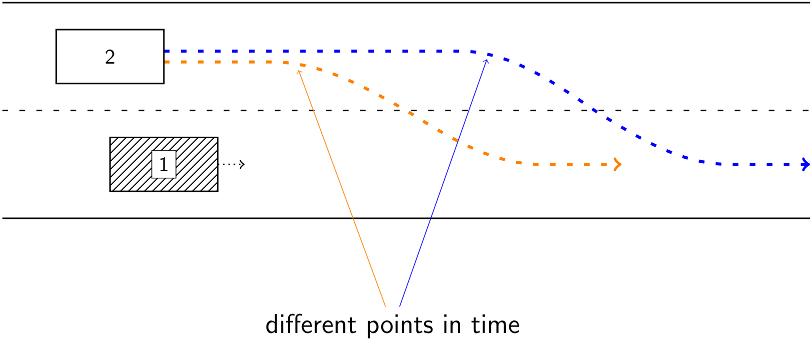
Semantics

What is needed to make it work?

Golog $Trans$ +

- Flexible timing
- Continuous change
- Multi-agent
- Robustness

Time and Continuous Change



Time and Continuous Change

From temporal sequential Golog:

$$time(A(\vec{x}, \tau)) = \tau$$

$$start(do(a, s)) = time(a)$$

From cc-Golog:

$\phi[s, \tau]$	evaluate ϕ in s at time τ
$\alpha[s, \tau]$	append new time parameter e.g. $jump[s, \tau] = jump(\tau)$

Time and Continuous Change

primitive action

$$
\begin{aligned}
Trans(\alpha, s, \delta, s') \equiv\ & \delta = Nil \wedge \\
& \exists \tau \,.\, \tau \geq start(s) \wedge \\
& Poss(\alpha[s, \tau], s) \wedge \\
& s' = do(\alpha[s, \tau], s)
\end{aligned}
$$

Time and Continuous Change

primitive action

monotonicity

$$
\begin{aligned}
Trans(\alpha, s, \delta, s') \equiv\ & \delta = Nil \land \\
& \exists \tau\,.\,\tau \geq start(s) \land \\
& Poss(\alpha[s, \tau], s) \land \\
& s' = do(\alpha[s, \tau], s)
\end{aligned}
$$

Time and Continuous Change

primitive action

monotonicity

$$
\begin{aligned}
Trans(\alpha, s, \delta, s') \equiv\ & \delta = Nil \wedge \\
& \exists \tau\,.\, \tau \geq start(s) \wedge \\
& Poss(\alpha[s, \tau], s) \wedge \\
& s' = do(\alpha[s, \tau], s)
\end{aligned}
$$

constrains τ further

Time and Continuous Change

primitive action

monotonicity

$$\begin{aligned} Trans(\alpha, s, \delta, s') \equiv \delta = Nil \wedge \\ \exists \tau \,.\, \tau \geq start(s) \wedge \\ Poss(\alpha[s, \tau], s) \wedge \\ s' = do(\alpha[s, \tau], s) \end{aligned}$$

constrains τ further

advance to time τ

Time and Continuous Change

primitive action

monotonicity

$$
\begin{aligned}
Trans(\alpha, s, \delta, s') \equiv\ & \delta = Nil \wedge \\
& \exists \tau \,.\, \tau \geq start(s) \wedge \\
& Poss(\alpha[s, \tau], s) \wedge \\
& s' = do(\alpha[s, \tau], s)
\end{aligned}
$$

constrains τ further

advance to time τ

$$Poss(waitFor(\phi, \tau), s) \equiv \phi[s, \tau]$$

Multiple Agents

Multi-agent: $\sigma_1 \parallel \ldots \parallel \sigma_n$ explains observations?

actor 1 actor n

Multiple Agents

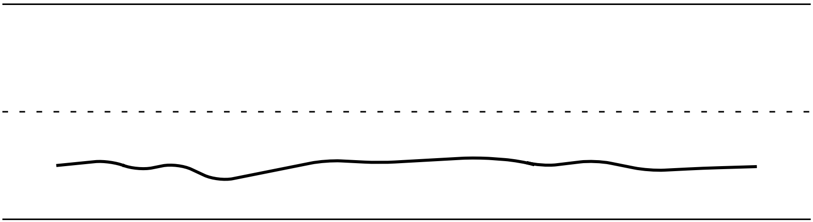
Multi-agent: $\sigma_1 \parallel \ldots \parallel \sigma_n$ explains observations?

actor 1 actor n

Concurrency as in ConGolog:

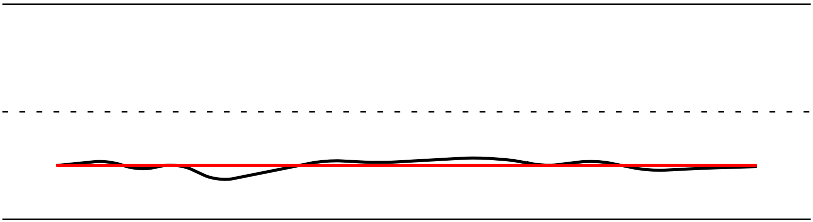
$$
\begin{aligned}
Trans(\sigma_1 \parallel \sigma_2, s, \delta, s') \equiv\ & \exists \delta' \,.\, Trans(\sigma_1, s, \delta', s') \land \delta = \delta' \parallel \sigma_2 \lor \\
& \exists \delta' \,.\, Trans(\sigma_2, s, \delta', s') \land \delta = \sigma_1 \parallel \delta'
\end{aligned}
$$

Robustness



Observed trace

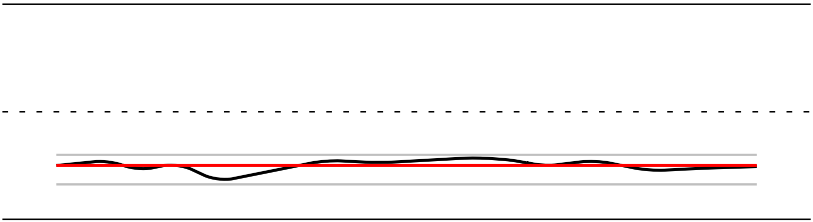
Robustness



Observed trace + model trace

Hypothesis: driving straight?

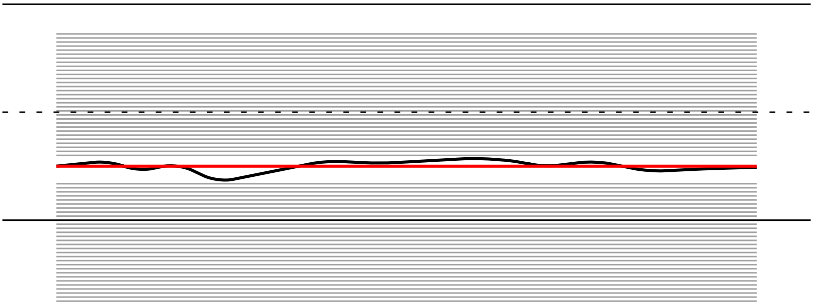
Robustness



Observed trace + model trace + lateral tolerance

Hypothesis: driving straight?

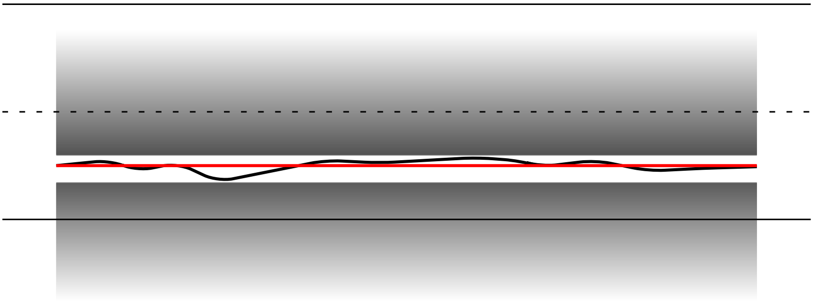
Robustness



Observed trace + model trace + lateral tolerances

Hypothesis: driving straight?

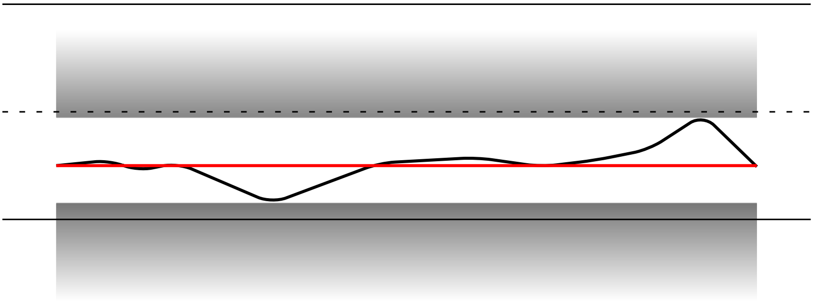
Robustness



Observed trace + model trace + weighted lateral tolerances

Hypothesis: driving straight? **Likely**

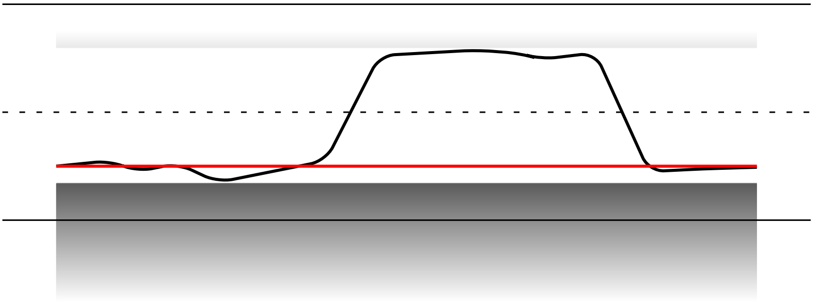
Robustness



Observed trace + model trace + weighted lateral tolerances

Hypothesis: driving straight? **Less likely**

Robustness



Observed trace + model trace + weighted lateral tolerances

Hypothesis: driving straight? **Unlikely**

Robustness

- Tolerances by stochastic actions
  $Choice(\beta, \alpha)$ and $prob_0(\beta, \alpha, s) \mapsto [0, 1]$

Robustness

- Tolerances by stochastic actions
  $Choice(\beta, \alpha)$ and $prob_0(\beta, \alpha, s) \mapsto [0, 1]$
- Rate situation by reward

$$r(s) \mapsto \mathbb{R}$$

user-supplied

Robustness

- Tolerances by stochastic actions
  $Choice(\beta, \alpha)$ and $prob_0(\beta, \alpha, s) \mapsto [0, 1]$
- Rate situation by reward

$$r(s) \mapsto \mathbb{R}$$

user-supplied

- Nondeterminism $\rightarrow$ choose best alternative

Robustness

- Tolerances by stochastic actions
  $Choice(\beta, \alpha)$ and $prob_0(\beta, \alpha, s) \mapsto [0, 1]$
- Rate situation by reward

$$r(s) \mapsto \mathbb{R}$$

user-supplied

- Nondeterminism $\rightarrow$ choose best alternative:
  1. Decompose σ into $(\gamma; \delta)$ *atomic* action
  2. Find best $(\gamma; \delta)$ amongst all decompositions
  3. Execute γ

Robustness

- Tolerances by stochastic actions
  $Choice(\beta, \alpha)$ and $prob_0(\beta, \alpha, s) \mapsto [0, 1]$
- Rate situation by reward

$$r(s) \mapsto \mathbb{R}$$

user-supplied

- Rate program by estimated reward

$$value(r, \sigma, s) \mapsto \mathbb{R}$$

- Nondeterminism $\rightarrow$ choose best alternative:
  1. Decompose σ into $(\gamma; \delta)$ *atomic* action
  2. Find best $(\gamma; \delta)$ amongst all decompositions
  3. Execute γ

Robustness: Decomposition

next atomic

input remainder

$$Next(\sigma, \gamma, \delta)$$

Robustness: Decomposition

next atomic

input remainder

$$Next(\sigma, \gamma, \delta)$$

Like $Trans$ without execution, e.g.:

$$Next(\alpha, \gamma, \delta) \equiv \gamma = \alpha \wedge \delta = Nil$$

$$Next(\sigma_1 \,|\, \sigma_2, \gamma, \delta) \equiv Next(\sigma_1, \gamma, \delta) \vee Next(\sigma_2, \gamma, \delta)$$

Robustness: Transition

$$
\begin{aligned}
& transPr(r,\sigma,s,\delta,s') = p \equiv \\
& \quad \textbf{if } \exists^1 \gamma_1,\delta_1 \,.\, Next(\sigma,\gamma_1,\delta_1) \land \\
& \quad\quad \big(\forall \gamma_2,\delta_2 \,.\, Next(\sigma,\gamma_2,\delta_2) \supset \\
& \quad\quad\quad value(r,(\gamma_1;\delta_1),s) \geq value(r,(\gamma_2;\delta_2),s)\big) \\
& \quad \textbf{then } \big(\textbf{if } \delta = \delta_1 \textbf{ then } p = transAtPr(r,\gamma_1,\delta_1,s,s') \textbf{ else } p = 0\big) \\
& \quad \textbf{else } p = 0
\end{aligned}
$$

decomposition $\gamma_1;\delta_1$ is optimal

execute γ_1

Robustness

Why decomposition? Decision theory + concurrency

$Trans$ recursively follows syntax tree
$\rightsquigarrow$ does not know "what comes after"

Robustness

Why decomposition? Decision theory + concurrency

$Trans$ recursively follows syntax tree
$\leadsto$ does not know "what comes after"

Program decomposition
$\leadsto$ full remaining program is always known
$\leadsto$ can resolve nondeterminism with remainder in mind

Robustness: Atomic Complex Actions

$$
\begin{aligned}
atomic(a; b) \parallel c \quad & \not\rightsquigarrow \quad do([a, c, b], S_0) \\
& \rightsquigarrow \quad do([a, b, c], S_0) \\
& \rightsquigarrow \quad do([c, a, b], S_0)
\end{aligned}
$$

Plan Recognition by Program Execution

Plan recognition. . .

- as satisfiability
- by iterative filtering of $allConsistPlans$
- by program execution

Plan Recognition by Program Execution

equivalent

Plan recognition. . .

- as satisfiability
- by iterative filtering of $allConsistPlans$
- by program execution

observe Actions

$$Poss(observe(\tau, \phi, \tau'), s) \equiv \tau = \tau' \wedge \phi[s, \tau]$$

Execution of $observe(\tau, \phi)$ means ϕ was observed at time τ

$observe$ Actions

$$Poss(observe(\tau, \phi, \tau'), s) \equiv \tau = \tau' \wedge \phi[s, \tau]$$

Execution of $observe(\tau, \phi)$ means ϕ was observed at time τ

$$(observe(\tau_1, \phi_1); \ldots; observe(\tau_n, \phi_n))$$

observe Actions

$$Poss(observe(\tau, \phi, \tau'), s) \equiv \tau = \tau' \wedge \phi[s, \tau]$$

Execution of $observe(\tau, \phi)$ means ϕ was observed at time τ

$$\sigma \qquad (observe(\tau_1, \phi_1); \ldots; observe(\tau_n, \phi_n))$$

observe Actions

$$Poss(observe(\tau, \phi, \tau'), s) \equiv \tau = \tau' \wedge \phi[s, \tau]$$

Execution of $observe(\tau, \phi)$ means ϕ was observed at time τ

$$\sigma \quad \| \quad (observe(\tau_1, \phi_1); \ldots; observe(\tau_n, \phi_n))$$

Online Heuristic

1. New observation (τ, ϕ) present:

$$\delta' = \delta \parallel observe(\tau, \phi)$$

merge observation

Online Heuristic

1. New observation (τ, ϕ) present:

$$\delta' = \delta \parallel observe(\tau, \phi)$$

merge observation

2. Enough *observe* actions buffered:

$$p' = p \cdot transPr(r, \delta, s, \delta', s')$$

resolves nondeterminism

Online Heuristic

1. New observation (τ, ϕ) present:

$$\delta' = \delta \parallel observe(\tau, \phi)$$

merge observation

2. Enough *observe* actions buffered:

$$p' = p \cdot transPr(r, \delta, s, \delta', s')$$

resolves nondeterminism

3. Reiterate.

Approach Summary

Model

proc $overtake(V, W)$
  $behind(V, W)?;$
  $leftLaneChange(V);$
  wait for $behind(W, V);$
  $rightLaneChange(V)$
$\dots$

Interpreter

Observations

at time 0: $pos(A) = (10, -2)$
at time 1: $pos(A) = (25, -2)$
at time 2: $pos(A) = (40, 0)$
$\dots$

Set of programs that
explain the observations.

Approach Summary

Candidate programs of the form

$$\sigma_1 \parallel \ldots \parallel \sigma_n$$

for n actors.

Model

proc $overtake(V, W)$
  $behind(V, W)?;$
  $leftLaneChange(V);$
  wait for $behind(W, V);$
  $rightLaneChange(V)$
$\ldots$

Interpreter

Set of programs that explain the observations.

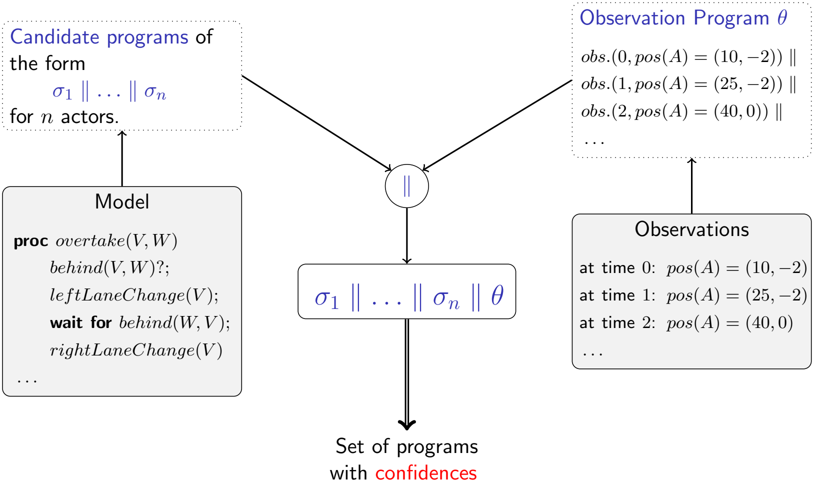
Observation Program θ

$obs.(0, pos(A) = (10, -2)) \parallel$
$obs.(1, pos(A) = (25, -2)) \parallel$
$obs.(2, pos(A) = (40, 0)) \parallel$
$\ldots$

Observations

at time 0: $pos(A) = (10, -2)$
at time 1: $pos(A) = (25, -2)$
at time 2: $pos(A) = (40, 0)$
$\ldots$

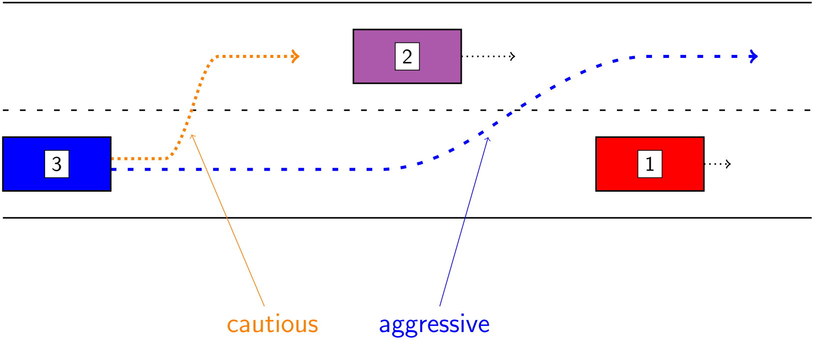
Approach Summary



Evaluation

- Prototype in ECLiPSe-CLP
- Sampling
- Linear constraint solver
  for equations from $waitFor$, $observe$

Demo



Video #1 Video #2

Conclusion

Accomplishments

- ✓ Flexible timing
- ✓ Continuous change
- ✓ Multi-agent
- ✓ Robustness
  - Model simplifies world

Conclusion

Plan Recognition by Program Execution

Accomplishments

- ✓ Flexible timing
- ✓ Continuous change
- ✓ Multi-agent
- ✓ Robustness
  - Model simplifies world
  - Sensor noise

Features

- Keeps it simple
- Sensor noise
- Efficient

Future Work

- Nonlinear constraints
- Extrapolate situation + remaining program

Appendix

Bibliography

Hung H. Bui, Svetha Venkatesh, and Geoff West. Policy recognition in the abstract hidden markov model. *Journal of Artificial Intelligence Research*, 17:2002, 2002.

Eugene Charniak and Robert Goldman. A probabilistic model of plan recognition. In *Proceedings of the ninth National conference on Artificial Intelligence*, volume 1 of *AAAI'91*, pages 160–165. AAAI Press, 1991.

Christopher Geib and Robert Goldman. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence*, 173: 1101–1132, 2009.

Alexandra Goultiaeva and Yves Lespérance. Incremental plan recognition in an agent programming framework. In *Cognitive Robotics Workshop*, pages 83–90, 2006.

Henry A. Kautz and James F. Allen. Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 32–37, 1986.

Miquel Ramirez and Hector Geffner. Plan recognition as planning. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

User Guarantees

The axiomizer must guarantee:

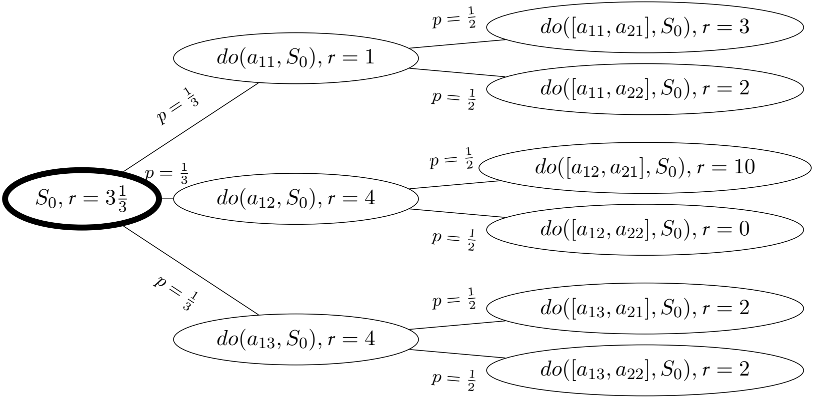
$$\mathcal{D} \models Choice(\beta, \alpha) \wedge (\exists \tau \,.\, \tau \geq start(s) \wedge Poss(\alpha[s, \tau], s)) \supset \\ prob_0(\beta, \alpha, s) > 0$$

$$\mathcal{D} \models (\exists \alpha \,.\, Choice(\beta, \alpha) \wedge \exists \tau \,.\, \tau \geq start(s) \wedge Poss(\alpha[s, \tau], s)) \supset \\ \sum_{\substack{\{\alpha \,|\, Choice(\beta,\alpha) \wedge \\ \exists \tau \,.\, \tau \geq start(s) \wedge \\ Poss(\alpha[s,\tau],s)\}}} prob_0(\beta, \alpha, s) = 1$$

$$\mathcal{D} \models \forall \beta \,.\, \exists f \,.\, \forall \alpha \,.\, Choice(\beta, \alpha) \supset (\exists i) f(i) = \alpha$$

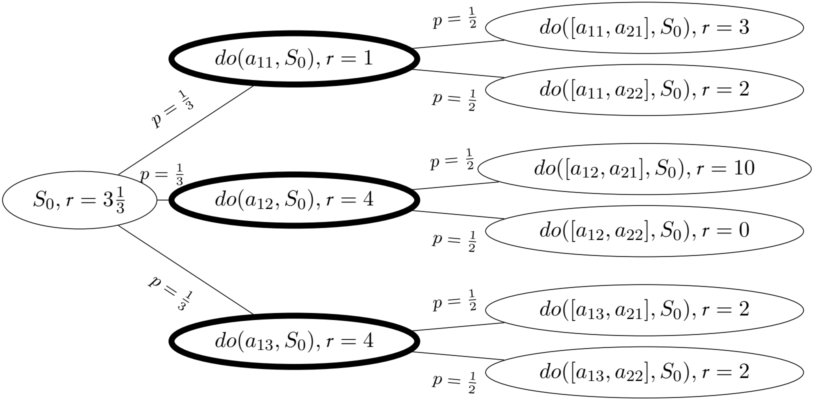
Robustness: *value*

$$value(r, \sigma, S_0) \geq 3\tfrac{1}{3}$$



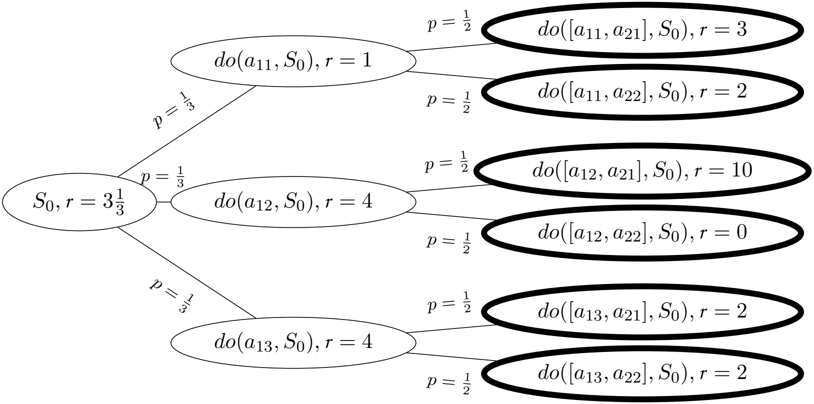
Robustness: *value*

$$value(r, \sigma, S_0) \geq 3$$



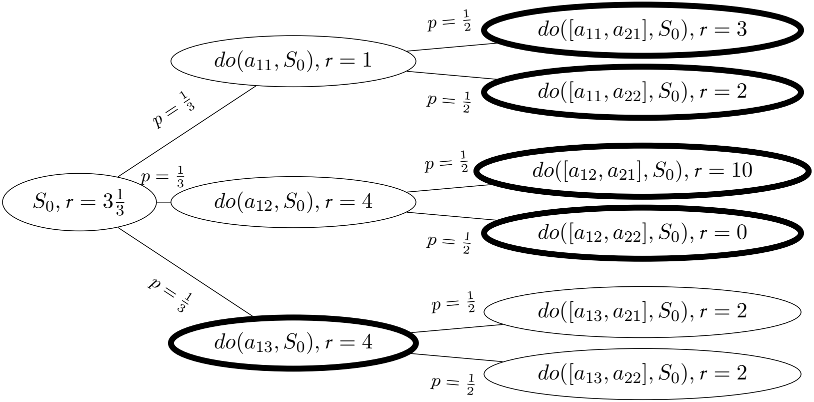
Robustness: *value*

$$value(r, \sigma, S_0) \geq 3\frac{1}{6}$$



Robustness: *value*

$$value(r, \sigma, S_0) = 3\frac{5}{6}$$



Robustness: *value*

$$Best(r,\sigma,s) \overset{\text{def}}{=} \forall P\,.\,\big(\forall s', s''\,.\,P(s') \wedge P(s'') \supset s' \not\sqsubset s''\big) \supset \sum_{\substack{\{(p,s') \,|\, \exists \delta\,.\,transPr^*(r,\sigma,s,\delta,s')=p\,\wedge \\ p>0 \,\wedge\, P(s')\}}} p \cdot r(s') \leq r(s)$$

$$value(r,\sigma,s) \overset{\text{def}}{=} \sum_{\substack{\{(p,s') \,|\, \exists \delta\,.\,transPr^*(r,\sigma,s,\delta,s')=p\,\wedge \\ p>0 \,\wedge\, Best(r,\delta,s')\,\wedge \\ \neg\exists s'',\delta\,.\,transPr^*(r,\sigma,s,\delta,s'')>0\,\wedge \\ Best(r,\delta,s'')\,\wedge\, s''\sqsubset s'\}}} p \cdot r(s')$$

Robustness: Sum Axiomatization

$$\sum_{\{\vec{x} \mid \Phi[\vec{X}/\vec{x}]\}} \nu(\vec{x})$$

$$\begin{aligned}
& sum_{\nu}(\Phi(\vec{X})) = v \stackrel{\text{def}}{=} \exists f, g \, . \\
& \quad (\forall \vec{x}) \big(\Phi[\vec{X}/\vec{x}] \supset (\exists i) \vec{x} = g(i) \big) \wedge \\
& \quad (\forall i, j) \big(\Phi[\vec{X}/g(i)] \wedge \Phi[\vec{X}/g(j)] \wedge i \neq j \supset g(i) \neq g(j) \big) \wedge \\
& \quad f(0) = 0 \wedge \\
& \quad (\forall i) \big((\Phi[\vec{X}/g(i)] \supset f(i+1) = f(i) + \nu(g(i))) \wedge \\
& \qquad \quad (\neg \Phi[\vec{X}/g(i)] \supset f(i+1) = f(i)) \big) \wedge \\
& \quad (\forall i) \big(f(i) \leq v \wedge \\
& \qquad \quad (\forall v')(f(i) \leq v' \supset v \leq v') \big)
\end{aligned}$$

Robustness: $Next$

$$
\begin{aligned}
Next(Nil, \gamma, \delta) &\equiv False \\
Next(\alpha, \gamma, \delta) &\equiv \gamma = \alpha \wedge \delta = Nil \\
Next(\beta, \gamma, \delta) &\equiv \gamma = \beta \wedge \delta = Nil \\
Next(\phi?, \gamma, \delta) &\equiv \gamma = \phi? \wedge \delta = Nil \\
Next(\pi v \,.\, \sigma, \gamma, \delta) &\equiv \exists x \,.\, Next(\sigma^v_x, \gamma, \delta) \\
Next(\sigma_1 \,|\, \sigma_2, \gamma, \delta) &\equiv Next(\sigma_1, \gamma, \delta) \vee Next(\sigma_2, \gamma, \delta) \\
Next(\sigma_1; \sigma_2, \gamma, \delta) &\equiv \exists \sigma_1' \,.\, Next(\sigma_1, \gamma, \sigma_1') \wedge \delta = \sigma_1'; \sigma_2 \vee \\
&\quad\; MaybeFinal(\sigma_1) \wedge Next(\sigma_2, \gamma, \delta) \\
Next(\sigma_1 \parallel \sigma_2, \gamma, \delta) &\equiv \exists \sigma_1' \,.\, Next(\sigma_1, \gamma, \sigma_1') \wedge \delta = \sigma_1' \parallel \sigma_2 \vee \\
&\quad\; \exists \sigma_2' \,.\, Next(\sigma_2, \gamma, \sigma_2') \wedge \delta = \sigma_1 \parallel \sigma_2' \\
Next(\sigma^*, \gamma, \delta) &\equiv \exists \sigma' \,.\, Next(\sigma, \gamma, \sigma') \wedge \delta = \sigma'; \sigma^*
\end{aligned}
$$

Robustness: $MaybeFinal$

$$
\begin{aligned}
MaybeFinal(Nil) &\equiv True \\
MaybeFinal(\alpha) &\equiv False \\
MaybeFinal(\beta) &\equiv False \\
MaybeFinal(\phi?) &\equiv False \\
MaybeFinal(\pi v \,.\, \sigma) &\equiv \exists x \,.\, MaybeFinal(\sigma^v_x) \\
MaybeFinal(\sigma_1 \mid \sigma_2) &\equiv MaybeFinal(\sigma_1) \vee MaybeFinal(\sigma_2) \\
MaybeFinal(\sigma_1 ; \sigma_2) &\equiv MaybeFinal(\sigma_1) \wedge MaybeFinal(\sigma_2) \\
MaybeFinal(\sigma_1 \parallel \sigma_2) &\equiv MaybeFinal(\sigma_1) \wedge MaybeFinal(\sigma_2) \\
MaybeFinal(\sigma^*) &\equiv True
\end{aligned}
$$

Robustness: $transAtPr$

$transAtPr(r, \alpha, \delta, s, s') = p \equiv$

if $\exists^1 \tau \,.\, \tau \geq start(s) \wedge Poss(\alpha[s,\tau], s) \wedge s' = do(\alpha[s,\tau], s)$

then $p = 1$ **else** $p = 0$

Robustness: $transAtPr$

rest program

$$transAtPr(r, \alpha, \delta, s, s') = p \equiv$$

$$\textbf{if } \exists^1 \tau \,.\, \tau \geq start(s) \wedge Poss(\alpha[s,\tau], s) \wedge s' = do(\alpha[s,\tau], s) \wedge$$
$$\big(\forall \tau', s'' \,.\, \tau' \geq start(s) \wedge Poss(\alpha[s,\tau'], s) \wedge s'' = do(\alpha[s,\tau'], s) \supset$$
$$value(r, \delta, s') \geq value(r, \delta, s'')\big)$$

choose r-maximizing τ

$$\textbf{then } p = 1 \textbf{ else } p = 0$$

Robustness: $transAtPr$

rest program

$$transAtPr(r, \alpha, \delta, s, s') = p \equiv$$

if $\exists^1 \tau \,.\, \tau \geq start(s) \wedge Poss(\alpha[s, \tau], s) \wedge s' = do(\alpha[s, \tau], s) \wedge$

$\big(\forall \tau', s'' \,.\, \tau' \geq start(s) \wedge Poss(\alpha[s, \tau'], s) \wedge s'' = do(\alpha[s, \tau'], s) \supset$

$value(r, \delta, s') \geq value(r, \delta, s'')\big)$ choose r-maximizing τ

then $p = 1$ **else** $p = 0$

$$transAtPr(r, \beta, \delta, s, s') = p \equiv$$

α outcome of β

if $\exists \alpha, p' \,.\, Choice(\beta, \alpha) \wedge$

prob. of outcome α

$transAtPr(r, \alpha, \delta, s, s') \cdot prob_0(\beta, \alpha, s) = p' \wedge p' > 0$

then $p = p'$ **else** $p = 0$

Robustness: $transAtPr$

rest program

$$transAtPr(r, \alpha, \delta, s, s') = p \equiv$$

if $\exists^1 \tau \,.\, \tau \geq start(s) \wedge Poss(\alpha[s, \tau], s) \wedge s' = do(\alpha[s, \tau], s) \wedge$

$(\forall \tau', s'' \,.\, \tau' \geq start(s) \wedge Poss(\alpha[s, \tau'], s) \wedge s'' = do(\alpha[s, \tau'], s) \supset$

$value(r, \delta, s') \geq value(r, \delta, s''))$

choose r-maximizing τ

then $p = 1$ **else** $p = 0$

$$transAtPr(r, \beta, \delta, s, s') = p \equiv$$

α outcome of β

if $\exists \alpha, p' \,.\, Choice(\beta, \alpha) \wedge$

prob. of outcome α

$transAtPr(r, \alpha, \delta, s, s') \cdot prob_0(\beta, \alpha, s) = p' \wedge p' > 0$

then $p = p'$ **else** $p = 0$

$$transAtPr(r, \phi?, \delta, s, s') = p \equiv$$

if $\phi[s] \wedge s' = s$ **then** $p = 1$ **else** $p = 0$.

Robustness: $transPr$

$$
\begin{aligned}
& transPr(r, \sigma, s, \delta, s') = p \equiv \\
& \quad \textbf{if } \exists^1 \gamma_1, \delta_1 \,.\, Next(\sigma, \gamma_1, \delta_1) \wedge \\
& \qquad \big(\forall \gamma_2, \delta_2 \,.\, Next(\sigma, \gamma_2, \delta_2) \supset \\
& \qquad\quad value(r, (\gamma_1; \delta_1), s) \geq value(r, (\gamma_2; \delta_2), s)\big) \\
& \quad \textbf{then } \big(\textbf{if } \delta = \delta_1 \textbf{ then } p = transAtPr(r, \gamma_1, \delta_1, s, s') \textbf{ else } p = 0\big) \\
& \quad \textbf{else } p = 0
\end{aligned}
$$

decomposition $\gamma_1; \delta_1$ is optimal

execute γ_1

Robustness: $transPr$ and $Trans$

different configurations

new semantics old semantics

$$\mathcal{D} \cup \mathcal{C} \cup \mathcal{C}' \models (\exists \delta, s') Trans(\sigma, s, \delta, s') \supset$$
$$(\exists \delta, s', p)(transPr(r, \sigma, s, \delta, s') = p \wedge$$
$$(p > 0 \vee r(s') = 0))$$

$$\mathcal{D} \cup \mathcal{C} \cup \mathcal{C}' \models transPr(r, \sigma, s, \delta, s') > 0 \supset Trans(\sigma, s, \delta, s')$$

same configurations

Robustness: $transPr^*$

$$
\begin{array}{l}
transPr^*(r, \sigma, s, \delta, s') = p \stackrel{\text{def}}{=} \\
\quad \textbf{if } \exists p' \,.\, \forall f \,.\, \big(\forall r', \sigma_1, s_0 \,.\, f(r', \sigma_1, s_0, \sigma_1, s_0) = 1\big) \wedge \\
\qquad\qquad \big(\forall r', \sigma_1, \delta_1, \delta_2, s_0, s_1, s_2, p_1, p_2 \,. \\
\qquad\qquad\quad p_1 > 0 \wedge f(r', \sigma_1, s_0, \delta_1, s_1) = p_1 \wedge \\
\qquad\qquad\quad p_2 > 0 \wedge transPr(r', \delta_1, s_1, \delta_2, s_2) = p_2 \supset \\
\qquad\qquad\qquad f(r', \sigma_1, s_0, \delta_2, s_2) = p_1 \cdot p_2\big) \supset \\
\qquad\qquad f(r, \sigma, s, \delta, s') = p' \\
\quad \textbf{then } p = p' \textbf{ else } p = 0
\end{array}
$$

Robustness: $Final$

$$
\begin{aligned}
Final(r, \sigma, s) \equiv\ & MaybeFinal(\sigma) \wedge \\
& value(r, Nil, s) \geq value(r, \sigma, s)
\end{aligned}
$$

Robustness: $doPr^*$

$$doPr(r, \sigma, s, s') = p \stackrel{\mathsf{def}}{=}$$

$$\textbf{if } \exists p' \,.\, transPr^*(r, \sigma, s, s') = p' \wedge Final(r, \sigma, s') \wedge$$

$$(\forall s'')\big(s \sqsubseteq s'' \wedge s'' \sqsubset s' \supset \neg Final(r, \sigma, s'')\big)$$

$$\textbf{then } p = p' \textbf{ else } p = 0$$

Atomic Complex Actions: Semantics

$$Next(atomic(\sigma), \gamma, \delta) \equiv \gamma = atomic(\sigma) \wedge \delta = Nil.$$

$$\begin{aligned} Next'(\sigma, \gamma, \delta) \stackrel{\text{def}}{=} \forall P \, . \, \big(&\forall \sigma', \gamma', \delta' \, . \, Next(\sigma', \gamma', \delta') \supset P(\sigma', \gamma', \delta')\big) \wedge \\ \big(&\forall \sigma', \sigma'', \gamma', \gamma'', \delta', \delta'' \, . \\ &\quad P(\sigma', \gamma', \delta') \wedge \gamma' = atomic(\sigma'') \wedge \\ &\quad Next(\sigma''; \delta', \gamma'', \delta'') \supset \\ &\qquad P(\sigma', \gamma'', \delta'')\big) \supset \\ &P(\sigma, \gamma, \delta) \wedge (\forall \sigma')\gamma \neq atomic(\sigma') \end{aligned}$$

Atomic Complex Actions: Plan Recognition

Candidate program:

- Make db inconsistent at $\tau_2 = 2$
- Regain consistency at $\tau_3 = 2$

Observations:

- $\tau_1 = 1$: $\phi_1 =$ "db cons."
- $\tau_2 = 2$: $\phi_2 =$ "db incons."
- $\tau_3 = 2$: $\phi_3 =$ "db cons."

Is (τ_2, ϕ_2) observable? No!

Inconsistent situation has timespan zero

Atomic Complex Actions: Plan Recognition

Candidate program:

- Make db inconsistent at $\tau_2 = 2$
- Regain consistency at $\tau_3 = 2$

Observations:

- $\tau_1 = 1$: $\phi_1 =$ "db cons."
- $\tau_2 = 2$: $\phi_2 =$ "db incons."
- $\tau_3 = 2$: $\phi_3 =$ "db cons."

Should $observe(\tau_2, \phi_2)$ be executable? No! But it is!

$$\begin{aligned}&\sigma \parallel (\ldots;\\ &\qquad atomic(observe(\tau_2, \phi_2); waitFor(now > \tau_2));\\ &\qquad \ldots)\end{aligned}$$