Augmenting Stream Constraint Programming with Eventuality Conditions

Jasper C.H. Lee [1] Jimmy H.M. Lee [2] Allen Z. Zhong [2]

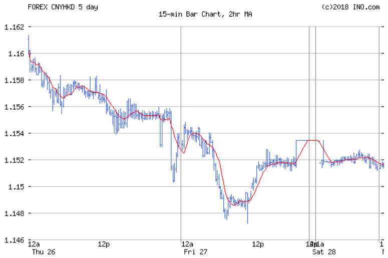
[1]Department of Computer Science
Brown University, Providence, RI 02912, USA

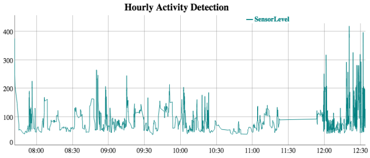
[2]Department of Computer Science and Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong

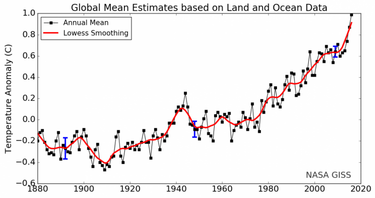
CP 2018, Lille, France

Introduction: Infinite Streams

Streams – infinite sequence over discrete time points







Introduction: Infinite Streams

Difficult to model streams in finite domain CSPs

Example

Want: equality between streams

$t = 0$	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$\cdots$
x_0	x_1	x_2	x_3	x_4	$\cdots$
y_0	y_1	y_2	y_3	y_4	$\cdots$

Need: $\forall t \geq 0$, x_t == y_t

Introduction: Stream CSPs

Siu et al. [IJCAI'11] and Lee and Lee [CP'14]: proposed framework, solving algorithm, and applications

Sequential Planning Problems

Real-time PID Controllers

Introduction: Stream CSPs

Stream Constraint Satisfaction

- Constraint programming on a new data type—streams
- Inherit all the benefits of declarative programming languages
  - Readability
  - Conciseness
  - Compositionality
  - Referential transparency

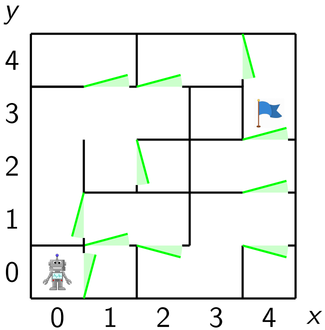
Introduction: Stream CSPs

Stream Constraint Satisfaction

- Constraint programming on a new data type—streams
- Inherit all the benefits of declarative programming languages
  - Readability
  - Conciseness
  - Compositionality
  - Referential transparency
- A natural CP formalism for modelling planning problems

Introduction: Research Issues

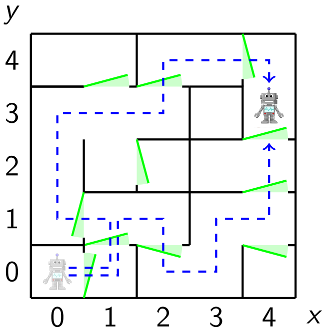
- Sequential planning [CP'14]



- Finite horizon, say, 12 steps

Introduction: Research Issues

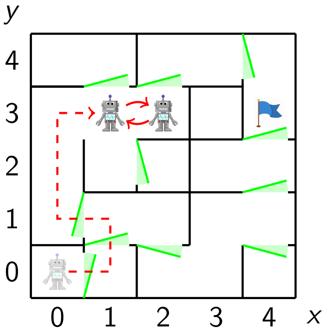
- Sequential planning [CP'14]



- `first next` ··· `next` *goal* == 1

Introduction: Research Issues

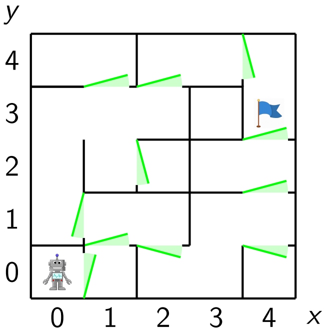
- Sequential planning [CP'14]



- ~~`first next` ··· `next` *goal* == 1~~

Introduction: Research Issues

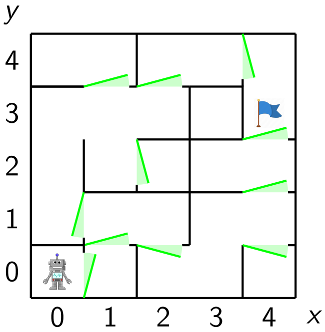
- Sequential planning [CP'14]



- *"Eventually"* achieving the goal?

Introduction: Research Issues

- Sequential planning [CP'14]



- Achieving the goal within *12 steps*?

Our Contributions

Two constructs for planning scenarios:

- the "Until" constraint for "eventuality" conditions
- an efficient syntactic sugar, @ operator, for conditions with hard deadline

Background: Infinite Streams

- Streams: function $\mathbb{N}_0 \to \Sigma$ (finite alphabet)

Example

$a = \langle 2, 3, 7, 2, 9, 4, 6, 5, \ldots \rangle$

	2	3	7	2	9	4	6	5	···
Time	0	1	2	3	4	5	6	7	···

- Stream variables: unknown streams

Background: Infinite Streams

- Operators
  - Pointwise: {+, -, *, /, %, and, or, if-then-else, ...}.

Example

$$\begin{array}{ccc} \langle 2,3,4,1,\ldots\rangle & & \langle\ 2\ \ 3\ \ 4\ \ 1\ \ \ldots\ \rangle \\ + & \equiv & \ \ +\ \ +\ \ +\ \ +\ \ \ldots \\ \langle 3,1,1,5,\ldots\rangle & & \langle\ 3\ \ 1\ \ 1\ \ 5\ \ \ldots\ \rangle \\ \hline & & \langle\ 5\ \ 4\ \ 5\ \ 6\ \ \ldots\ \rangle \end{array}$$

Background: Infinite Streams

- Operators
  - Pointwise: {+, -, *, /, %, and, or, if-then-else, ...}.
  - Temporal: {first, next, fby}.

Example

- First (first)
  $\texttt{first}\ \langle 1, 2, 3, 4, \ldots \rangle = \langle 1, 1, 1, 1, \ldots \rangle$
- Next (next)
  $\texttt{next}\ \langle 1, 2, 3, 4, \ldots \rangle = \langle 2, 3, 4, \ldots \rangle$

Background: Infinite Streams

- Operators
  - Pointwise: {`+`, `-`, `*`, `/`, `%`, `and`, `or`, `if-then-else`, ...}.
  - Temporal: {`first`, `next`, `fby`}.

Example

- First (`first`)
  $\texttt{first}\ \langle 1, 2, 3, 4, \ldots \rangle = \langle 1, 1, 1, 1, \ldots \rangle$
- Next (`next`)
  $\texttt{next}\ \langle 1, 2, 3, 4, \ldots \rangle = \langle 2, 3, 4, \ldots \rangle$
- $\texttt{first next}\ \langle 1, 2, 3, 4, \ldots \rangle = \langle 2, 2, 2, 2, \ldots \rangle$

Background: Stream Constraints

- (Pointwise) Stream constraints: relations between streams, e.g. pointwise arithmetic comparison {<, <=, ==, !=, >=, >}

Example

constraint $\texttt{first}\ A + \texttt{next}\ B > C$ satisfied!

$$
\begin{aligned}
A &= \langle 1, 2, 4, 3, 2, 6, 4, 2, \ldots \rangle \\
B &= \langle 3, 5, 1, 4, 1, 1, 3, 2, \ldots \rangle \\
\texttt{first}\ A + \texttt{next}\ B &= \langle 6, 2, 5, 2, 2, 4, 3, \ldots \rangle \\
C &= \langle 1, 1, 4, 1, 0, 3, 1, \ldots \rangle
\end{aligned}
$$

Background: Stream Constraints

- (Pointwise) Stream constraints: relations between streams, e.g. pointwise arithmetic comparison $\{<, <=, ==, !=, >=, >\}$

Example

constraint $\texttt{first}\ A + \texttt{next}\ B > C$ not satisfied!

$$\begin{aligned} A &= \langle 1, 2, 1, 3, 2, 6, 4, 2, \ldots \rangle \\ B &= \langle 3, 5, 1, 4, 1, 1, 3, 2, \ldots \rangle \\ \texttt{first}\ A + \texttt{next}\ B &= \langle 6, 2, 5, 2, 2, 4, 3, \ldots \rangle \\ C &= \langle 1, 1, 6, 1, 1, 1, 1, \ldots \rangle \end{aligned}$$

Background: Stream CSPs

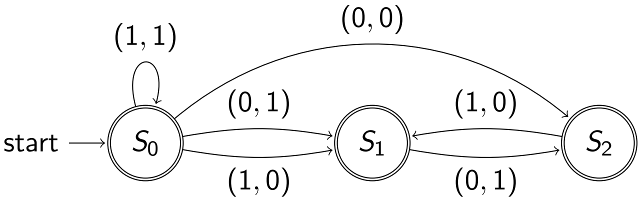
A *Stream Constraint Satisfaction Problem* (St-CSP) $P = (X, D, C)$

- X: a set of stream variables/unknowns
- D: a function mapping each variable to its domain, which is the set of all streams with the variable's alphabet
- C: a set of stream constraints to restrict combination of stream values that the variables can take

A *solution* to the St-CSP P is a consistent variable assignment to all variables to that all constraints are satisfied simultaneously

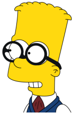
Background: Stream CSPs

- *Sol(P)*: *deterministic ω-regular language*
- Representation: *deterministic Büchi automaton $\mathcal{A}$*



- Run: $S_0 \xrightarrow{(1,1)} S_0 \xrightarrow{(0,1)} S_1 \xrightarrow{(0,1)} S_2 \xrightarrow{(1,0)} \ldots$
- Solution streams: $\langle 1, 0, 0, 1, \ldots \rangle$ and $\langle 1, 1, 1, 0, \ldots \rangle$

Example St-CSP

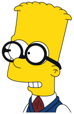


Not read

Not read

Not read

Example St-CSP

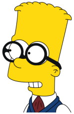


Reading...

Not read

Not read

Example St-CSP

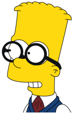


Done

Reading...

Not read

Example St-CSP

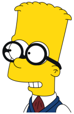


Done

Done

Reading...

Example St-CSP

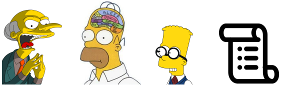


Done

Done

Done

Example St-CSP



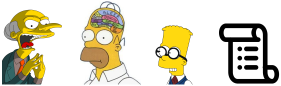
$giveTo : [0, 2];\ read_0, read_1, read_2, goal : [0, 1];$

For each i,

$\texttt{first}\ read_i == 0;$

$\texttt{next}\ read_i == read_i\ \texttt{or}\ (giveTo\ \texttt{eq}\ i);$

Example St-CSP



$$\mathit{giveTo} : [0, 2];\ \mathit{read}_0, \mathit{read}_1, \mathit{read}_2, \mathit{goal} : [0, 1];$$

$$\begin{aligned}
&\textit{For each } i, \\
&\quad \texttt{first } \mathit{read}_i == 0; \\
&\quad \texttt{next } \mathit{read}_i == \mathit{read}_i \texttt{ or } (\mathit{giveTo} \texttt{ eq } i); \\
&\mathit{goal} == (\mathit{read}_0 \texttt{ and } \mathit{read}_1 \texttt{ and } \mathit{read}_2);
\end{aligned}$$

Eventuality Condition

- Guarantee eventual success without imposing finite horizon?
- No, all constraints are inherently pointwise

Eventuality Condition

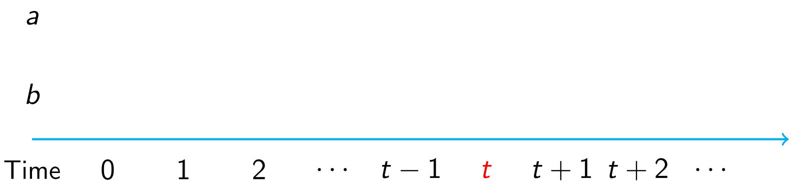
- Guarantee eventual success without imposing finite horizon?
- No, all constraints are inherently pointwise
  - Formal proof (not in paper): straightforward adaptation of the finite automata pumping lemma

The "Until" Constraint

Definition (a `until` b)

$\exists t \geq 0$, s.t.

-

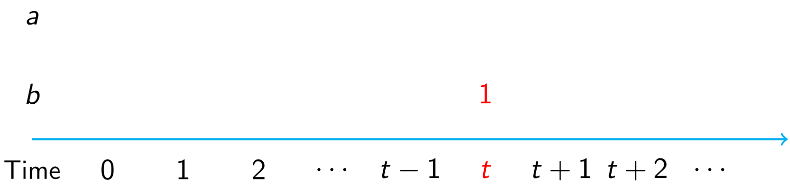


The “Until” Constraint

Definition (a `until` b)

$\exists t \geq 0$, s.t.

- $b(t) \neq 0$ (eventually)



The “Until” Constraint

Definition (a `until` b)

$\exists t \geq 0$, s.t.

- $b(t) \neq 0$ (eventually)

a									
b	0	0	0	$\cdots$	0	1			
Time	0	1	2	$\cdots$	$t-1$	t	$t+1$	$t+2$	$\cdots$

The "Until" Constraint

Definition (a `until` b)

$\exists t \geq 0$, s.t.

- $b(t) \neq 0$ (eventually)
- $\forall j < t,\ a(j) \neq 0$

a	1	2	1	$\cdots$	3				
b	0	0	0	$\cdots$	0	1			
Time	0	1	2	$\cdots$	$t-1$	t	$t+1$	$t+2$	$\cdots$

The "Until" Constraint

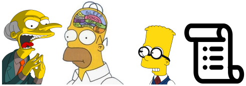
Definition (a `until` b)

$\exists t \geq 0$, s.t.

- $b(t) \neq 0$ (eventually)
- $\forall j < t,\ a(j) \neq 0$

a	1	2	1	$\cdots$	3	0	0	1	$\cdots$
b	0	0	0	$\cdots$	0	1	2	0	$\cdots$
Time	0	1	2	$\cdots$	$t-1$	t	$t+1$	$t+2$	$\cdots$

The "Until" Constraint



$giveTo : [0, 2];\ read_0, read_1, read_2, goal : [0, 1];$

For each i,

  $\texttt{first}\ read_i == 0;$

  $\texttt{next}\ read_i == read_i\ \texttt{or}\ (giveTo\ \mathsf{eq}\ i);$

$goal == (read_0\ \texttt{and}\ \cdots\ \texttt{and}\ read_2);$

$1\ \texttt{until}\ (goal\ \mathsf{eq}\ 1);$

The @ Operator: Problem

- How did we enforce that the circulation finishes within 10 steps?

The @ Operator: Problem

- How did we enforce that the circulation finishes within 10 steps?
- "first $\underbrace{\texttt{next} \cdots \texttt{next}}_{10\ \texttt{next}\ \text{operators}}$ *goal* == 1"

The @ Operator: Problem

- How did we enforce that the circulation finishes within 10 steps?
- "$\texttt{first}\ \underbrace{\texttt{next}\ \cdots\ \texttt{next}}_{10\ \texttt{next}\ \text{operators}}\ \textit{goal} == 1$"
- It is cumbersome and time inefficient

The @ Operator

Definition ($x@t$)

$\forall i \geq 0,\ (x@t)(i) = x(t)$

x :	2	3	7	$\cdots$	9	4	6	$\cdots$
Time	0	1	2	$\cdots$	$t-1$	t	$t+1$	$\cdots$

The @ Operator

Definition ($x@t$)

$\forall i \geq 0, (x@t)(i) = x(t)$

x :	2	3	7	$\cdots$	9	4	6	$\cdots$
$x@t$:	4	4	4	$\cdots$	4	4	4	$\cdots$
Time	0	1	2	$\cdots$	$t-1$	t	$t+1$	$\cdots$

The @ Operator

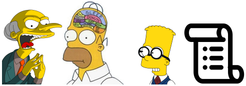
Definition ($x@t$)

$\forall i \geq 0,\ (x@t)(i) = x(t)$

x :	2	3	7	$\cdots$	9	4	6	$\cdots$
$x@t$:	4	4	4	$\cdots$	4	4	4	$\cdots$
Time	0	1	2	$\cdots$	$t-1$	t	$t+1$	$\cdots$

Equivalent form: "$\texttt{first}\ \underbrace{\texttt{next}\ \cdots\ \texttt{next}}_{t\ \texttt{next}\ \text{operators}}\ x$"

The @ Operator



$giveTo : [0, 3];\ read_0, \ldots, read_2, goal : [0, 1];$

For each i,

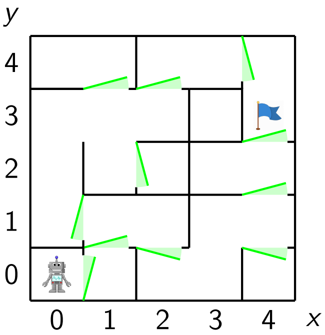
first $read_i$ == 0;

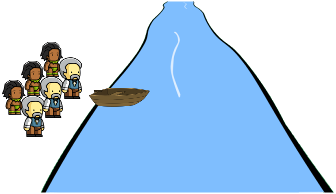
next $read_i$ == $read_i$ or ($giveTo$ eq i);

$goal$ == ($read_0$ and $\cdots$ and $read_2$);

$goal$ @10 == 1;

Experiments

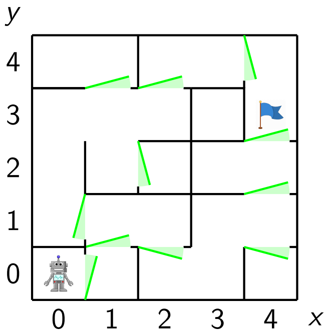




Experiments: Path Planning

Experiment settings:

- $n \times n$ grid world
- directed edges (probability p)
- 50 random instances
- timeout: 600 seconds



Experiments: Path Planning

Q1: Is there a plan for the robot to reach the goal eventually?

- St-CSP: 1 `until` (*goal* eq 1)
- Finite domain CSP (single solution): increasing horizon using Gecode v6.0.0
- Finite domain CSP (single solution): fix max length n^2

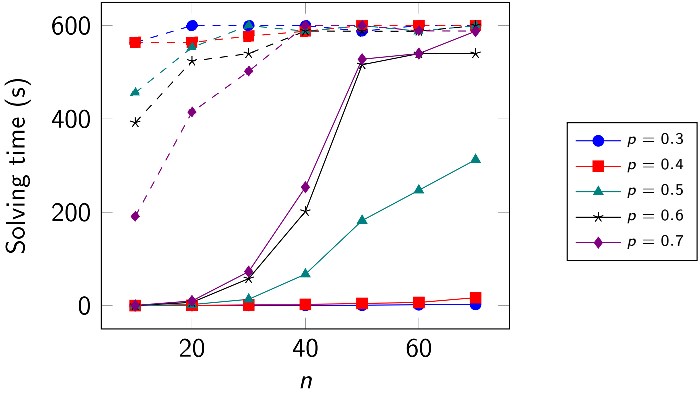
Experiments: Path Planning

Q1: Is there a plan for the robot to reach the goal eventually?

- St-CSP: 1 `until` (*goal* eq 1)
- Finite domain CSP (single solution): increasing horizon using Gecode v6.0.0
- Finite domain CSP (single solution): fix max length n^2
  - Memory issue at $n = 40$, exceeding 256G ($O(n^2)$ variables)
  - Many instances already timed out at $n = 10$

Experiments: Path Planning

- St-CSP with "`until`" (Solid)
- Finite domain CSP, increasing horizon (Dashed)



Experiments: Path Planning

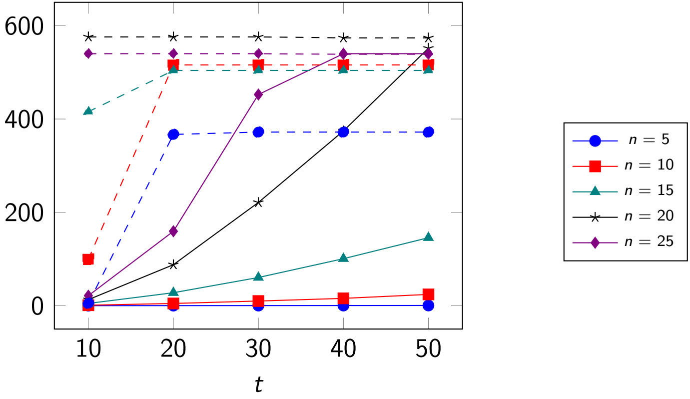
Q2: Is there a plan for the robot to reach the goal within t steps?

- St-CSP: $\texttt{first}\ \underbrace{\texttt{next}\ \cdots\ \texttt{next}}_{t\ \texttt{next}\ \text{operators}}\ \textit{goal} == 1$
- St-CSP: $\textit{goal}\ \texttt{@}\ t == 1$
- Finite domain CSP using Gecode v6.0.0 (single solution)

Experiments: Path Planning

St-CSP: `first next` (dashed) vs `@` (solid)

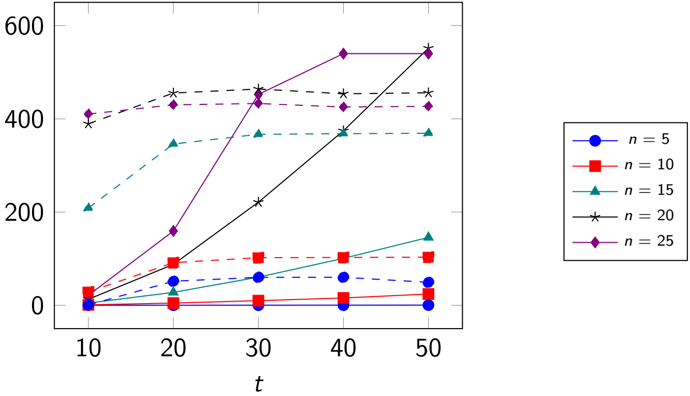
$p = 0.8$



Experiments: Path Planning

Finite domain CSP (dashed) vs St-CSP with @ (solid)

$p = 0.8$



Concluding Remarks

- Enhanced the expressiveness of St-CSP framework
  - The `Until` constraint: eventuality condition
  - The `@` operator: improved modelling and efficiency for goal condition with explicit deadline

Concluding Remarks

- Enhanced the expressiveness of St-CSP framework
  - The `Until` constraint: eventuality condition
  - The `@` operator: improved modelling and efficiency for goal condition with explicit deadline
  - Natural formalism: no need for increasing horizon

Concluding Remarks

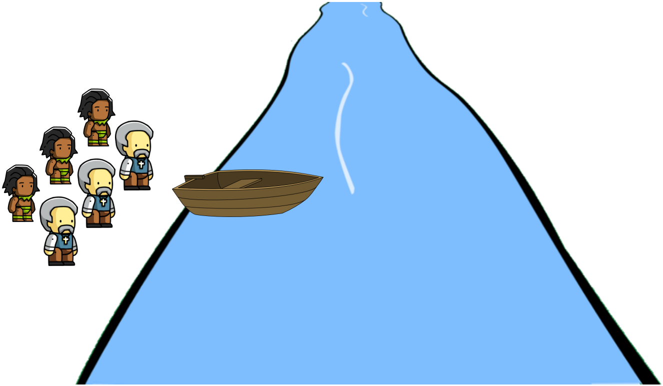
- Enhanced the expressiveness of St-CSP framework
  - The `Until` constraint: eventuality condition
  - The `@` operator: improved modelling and efficiency for goal condition with explicit deadline
  - Natural formalism: no need for increasing horizon
  - Towards a bridge between CP and Planning

Future Work

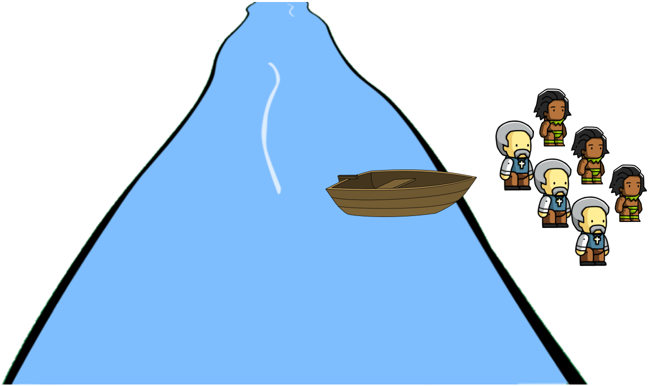
- Solving algorithm for single solution
- Generalize $x@t$ so that t can be a variable
- Stream constraint optimization
- Correspondence between St-CSP and Planning
- Adversarial planning (safety+reachability games)

Thank you!!

Conclusion: Missionaries and Cannibals



Conclusion: Missionaries and Cannibals



Conclusion: Missionaries and Cannibals

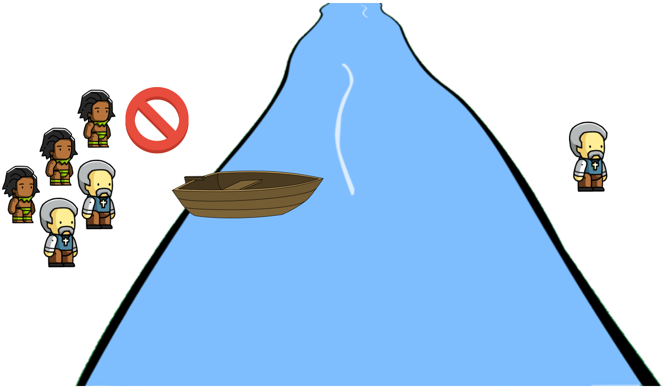
$M_{left}, M_{right}, C_{left}, C_{right}$: [0, 3]; $boat, goal$: [0, 1];

// initial conditions
`first` M_{left} == 3; `first` C_{left} == 3;
`first` M_{right} == 0; `first` C_{right} == 0;
`first` $boat$ == 0;

// finish the game when everyone is on the other bank
$goal$ == M_{right} eq 3 and C_{right} eq 3;

// stop moving people once the goal is achieved
$goal$ -> (`next` M_{left}) eq M_{left};
$goal$ -> (`next` C_{left}) eq C_{left};

Conclusion: Missionaries and Cannibals

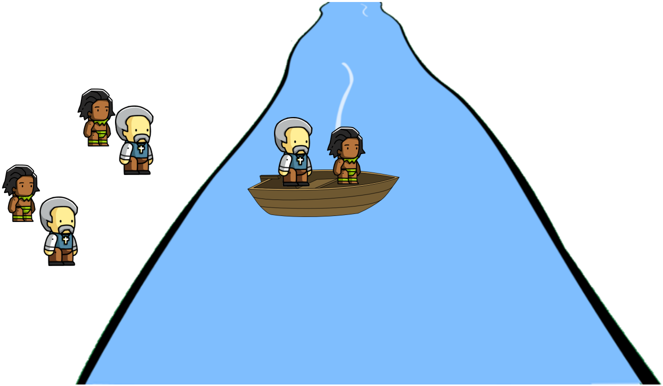


Conclusion: Missionaries and Cannibals

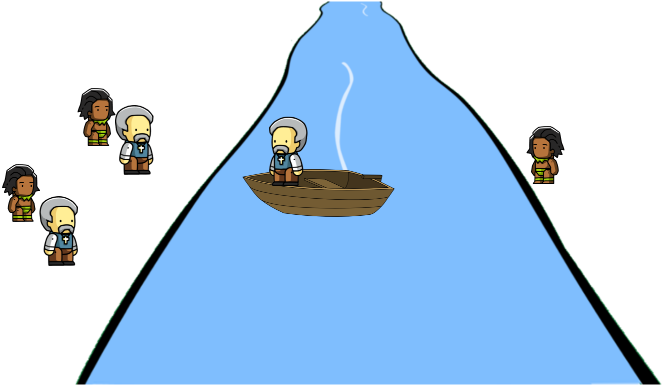
$M_{left}, M_{right}, C_{left}, C_{right} : [0, 3];$ $boat, goal : [0, 1];$

// on each bank, cannibals cannot outnumber missonaries
C_{left} `<=` `if` M_{left} `eq` 0 `then` 3 `else` M_{left};
C_{right} `<=` `if` M_{right} `eq` 0 `then` 3 `else` M_{right};

Conclusion: Missionaries and Cannibals



Conclusion: Missionaries and Cannibals



Conclusion: Missionaries and Cannibals

$M_{left}, M_{right}, C_{left}, C_{right} : [0,3];\ boat, goal : [0,1];$

// The boat needs at least 1 person until we finish the game

$\texttt{abs}(M_{left} - \texttt{next}\ M_{left}) + \texttt{abs}(C_{left} - \texttt{next}\ C_{left}) >= \texttt{not}(goal);$

// The boat has capacity 2

$\texttt{abs}(M_{left} - \texttt{next}\ M_{left}) + \texttt{abs}(C_{left} - \texttt{next}\ C_{left}) <= 2;$

// The direction of the boat always alternates until we finish the game

$\texttt{next}\ boat == \texttt{if}\ goal\ \texttt{then}\ boat\ \texttt{else}\ \texttt{not}(boat);$

Conclusion: Missionaries and Cannibals

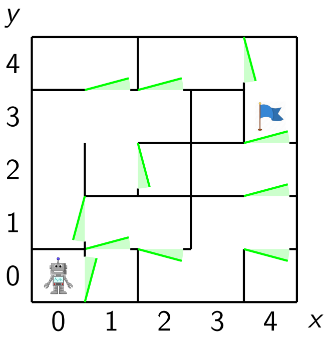
$M_{left}, M_{right}, C_{left}, C_{right} : [0, 3]$; $boat, goal : [0, 1]$;

// Axiom 1: Conservation of mass
M_{left} + M_{right} == 3;
C_{left} + C_{right} == 3;

// Axiom 2: The direction of the boat determines the moves of mass
boat eq 1 -> (M_{left} - `next` M_{left}) `le` 0;
boat eq 1 -> (C_{left} - `next` C_{left}) `le` 0;
boat eq 0 -> (M_{left} - `next` M_{left}) ge 0;
boat eq 0 -> (C_{left} - `next` C_{left}) ge 0;

Conclusion: Path Planning

- The robot wants to reach the goal starting from some point.



$x, y : [0, 4]$; $goal : [0, 1]$

`first` x == 0;

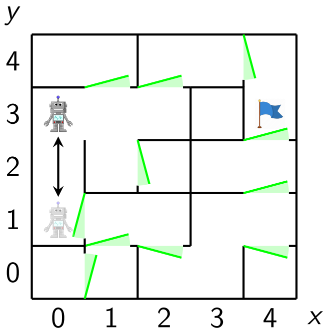
`first` y == 0;

`first` $goal$ == 0;

`next` $goal$ == (x `eq` 4 `and` y `eq` 3) `or` $goal$;

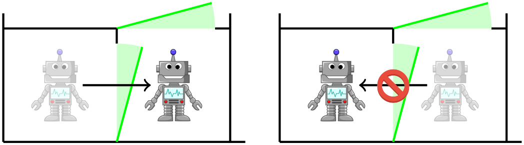
Conclusion: Path Planning

- The robot can move around if there are no blocking walls or doors.



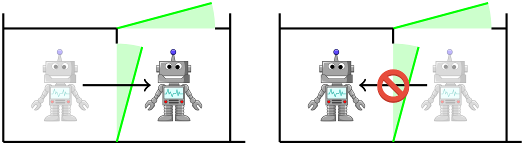
Conclusion: Path Planning

- The door is 1-way!



Conclusion: Path Planning

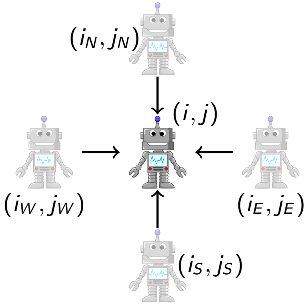
- The door is 1-way!



- The maze forms a directed graph.

Conclusion: Path Planning

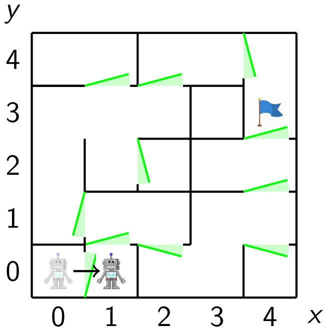
The robot is in cell (i, j) if it stays there from the last time point or moves from any of (i_E, j_E), (i_S, j_S), (i_W, j_W) or (i_N, j_N).



((`next` x eq i) and (`next` y eq j)) -> (x eq i and y eq j)
or (x eq i_E and y eq j_E) or $\cdots$ or (x eq i_N and y eq j_N);

Conclusion: Path Planning

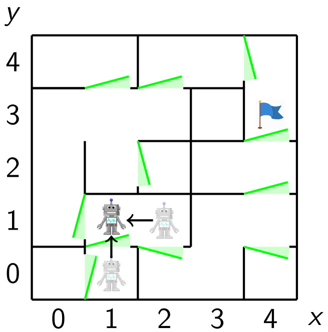
The robot is in cell $(1, 0)$ if it stays there from the last time point or moves from $(0, 0)$.



((next x eq 1) and (next y eq 0)) -> (x eq 1 and y eq 0)
or (x eq 0 and y eq 0);

Conclusion: Path Planning

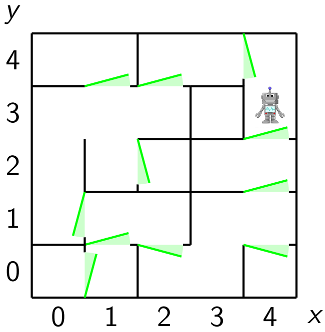
The robot is in cell $(1, 1)$ if it stays there from the last time point or moves from $(1, 0)$ or $(2, 1)$.



((next x eq 1) and (next y eq 1)) -> (x eq 1 and y eq 1)
or (x eq 1 and y eq 0) or (x eq 2 and y eq 1);

Conclusion: Path Planning

- The robot stops moving once reaching the goal.



goal –> ((*x* eq `next` *x*) and (*y* eq `next` *y*));

Experiments: Missionaries and Cannibals

Q1: Is there a safe plan that *eventually* moves everyone over?

- Settings
  - $n = 40 - 240$: number of missionaries/cannibals
  - $b = 4 - 8$: capacity of boat
  - 600 seconds timeout
- St-CSP: 1 `until` (*goal* eq 1)
- Finite domain CSP (single solution): increasing horizon (Gecode v6.0.0)

- Finite domain CSP (single solution): fix max length $n(b+1)$

Experiments: Missionaries and Cannibals

Q1: Is there a safe plan that *eventually* moves everyone over?

- Settings
  - $n = 40 - 240$: number of missionaries/cannibals
  - $b = 4 - 8$: capacity of boat
  - 600 seconds timeout
- St-CSP: 1 `until` (*goal* eq 1)
- Finite domain CSP (single solution): increasing horizon (Gecode v6.0.0)
  - Timed out in all tested instances.
- Finite domain CSP (single solution): fix max length $n(b+1)$

Experiments: Missionaries and Cannibals

Q1: Is there a safe plan that *eventually* moves everyone over?

- Settings
  - $n = 40 - 240$: number of missionaries/cannibals
  - $b = 4 - 8$: capacity of boat
  - 600 seconds timeout
- St-CSP: 1 `until` (*goal* eq 1)
- Finite domain CSP (single solution): increasing horizon (Gecode v6.0.0)
  - Timed out in all tested instances.
- Finite domain CSP (single solution): fix max length $n(b+1)$
  - All solved within 15 seconds but ...

Experiments: Missionaries and Cannibals

Q1: Is there a safe plan that *eventually* moves everyone over?

- Settings
  - $n = 40 - 240$: number of missionaries/cannibals
  - $b = 4 - 8$: capacity of boat
  - 600 seconds timeout
- St-CSP: 1 `until` (*goal* eq 1)

	$b = 4$	$b = 5$	$b = 6$	$b = 7$	$b = 8$
$n = 40$	1.456	1.939	2.307	2.537	2.959
$n = 80$	9.979	13.45	17.324	21.356	26.229
$n = 120$	33.56	44.782	59.113	73.335	91.351
$n = 160$	76.532	105.341	139.212	175.149	219.134
$n = 200$	150.137	207.466	274.537	348.243	436.469
$n = 240$	259.773	360.413	474.005	–	–

Experiments: Missionaries and Cannibals

Q2: Is there a safe plan that moves everyone over within t steps?

- St-CSP: $goal$ `@` t `==` 1 vs `first` $\underbrace{\texttt{next} \cdots \texttt{next}}_{t \texttt{ next} \text{ operators}}$ $goal$

(n, b)	$t = 10$	$t = 40$	$t = 70$	$t = 100$
$(20, 5)$	0.64/49.68	4.04/–	9.21/–	14.84/–
$(30, 6)$	1.71/178.68	16.33/–	36.23/–	56.76/–
$(40, 7)$	4.01/454.98	38.55/–	95.19/–	152.79/–
$(50, 8)$	9.07/–	100.34/–	236.58/–	374.07/–
$(60, 9)$	17.31/–	183.89/–	461.51/–	–/–
$(70, 10)$	32.25/–	371.57/–	–/–	–/–

Experiments: Missionaries and Cannibals

Q2: Is there a safe plan that moves everyone over within t steps?

- Finite domain CSP using Gecode v6.0.0 (single solution)

(n, b)	$t = 10$	$t = 40$	$t = 70$	$t = 100$
$(20, 5)$	0.663	0.435	0.562	1.075
$(30, 6)$	0.435	0.560	0.780	1.011
$(40, 7)$	0.562	0.519	0.799	1.139
$(50, 8)$	0.762	0.521	0.767	1.102
$(60, 9)$	1.002	0.501	0.835	0.975
$(70, 10)$	1.425	0.526	0.873	0.1109

Experiments: Missionaries and Cannibals

Q2: Is there a safe plan that moves everyone over within t steps?

- Finite domain CSP using Gecode v6.0.0 (single solution) but most timed out when searching for all solutions ...

(n, b)	$t = 10$	$t = 40$	$t = 70$	$t = 100$
$(20, 5)$	0.663	0.435	0.562	1.075
$(30, 6)$	0.435	0.560	0.780	1.011
$(40, 7)$	0.562	0.519	0.799	1.139
$(50, 8)$	0.762	0.521	0.767	1.102
$(60, 9)$	1.002	0.501	0.835	0.975
$(70, 10)$	1.425	0.526	0.873	0.1109

Appendix: Solving Algorithm

High level approach of Lee and Lee [CP'14]:

$$\text{St-CSP} \xrightarrow{\textit{Normalise}} \text{Normalised St-CSP} \xrightarrow{\textit{Solve}} \text{Büchi Automaton}$$

Appendix: Solving Algorithm

St-CSP $\xrightarrow{\textit{Normalise}}$ Normalised St-CSP $\xrightarrow{\textit{Solve}}$ Büchi Automaton

- Primitive next constraints: $x_i == \texttt{next}\ x_j$
- Primitive pointwise constraints with no next or fby (but can contain first operators)

Appendix: Solving Algorithm

St-CSP $\xrightarrow{Normalise}$ Normalised St-CSP $\xrightarrow{Solve}$ Büchi Automaton

- Primitive next constraints: $x_i == \texttt{next}\ x_j$
- Primitive pointwise constraints with no next, fby or until (but can contain first operators)
- Primitive until constraints: $x_i\ \texttt{until}\ x_j$
- Primitive @ constraints: x_i == x_j @t, where $t \geq 1$

Appendix: Solving Algorithm

High level approach of Lee and Lee [CP'14]:

$$\text{St-CSP} \xrightarrow{\textit{Normalise}} \text{Normalised St-CSP} \xrightarrow{\textit{Solve}} \text{Büchi Automaton}$$

- depth first search
- stream: infinite size
  - variables cannot be instantiated with stream values completely
  - instantiating the values in the order of time point

Appendix: Solving Algorithm

High level approach of Lee and Lee [CP'14]:

$$\text{St-CSP} \xrightarrow{\textit{Normalise}} \text{Normalised St-CSP} \xrightarrow{\textit{Solve}} \text{Büchi Automaton}$$

Solving x_i `until` x_j:

- If $x_j \neq 1$, add "x_i `until` x_j" in the new constraint set
- If $x_j = 1$, do not add "x_i `until` x_j" in the new constraint set

Appendix: Solving Algorithm

High level approach of Lee and Lee [CP'14]:

St-CSP $\xrightarrow{\textit{Normalise}}$ Normalised St-CSP $\xrightarrow{\textit{Solve}}$ Büchi Automaton

Solving x_i == x_j @ t:

- If $t > 1$, then we include "x_i == x_j @ $(t - 1)$ in the new constraint set
- If $t = 1$, then we include "x_i == `first` x_j in the new constraint set