Computing Applicability Conditions for Plans with Loops

Siddharth Srivastava  Neil Immerman  Shlomo Zilberstein

Department of Computer Science, University of Massachusetts Amherst

Twentieth International Conference on Automated Planning and Scheduling
15 May, 2010
Abacus Programs Translation Examples

Plans with Loops

Loops = ⇒ smaller, more general plans; allow unknown quantities

Siddharth Srivastava

Applicability Conditions for Plans with Loops
Loops $\implies$ smaller, more general plans; allow unknown quantities
Correct numbers of iterations can reach any point in the interior

[Bonet et al., 2009]
Correct numbers of iterations can reach any point in the interior
Abacus Programs Translation Examples

Plans with Loops: Risk

Never reaches G!

Applicability Conditions for Plans with Loops
A plan with loops may traverse almost the entire state space – without solving the problem.

Never reaches G!
Why analyze loops of actions?

- During construction: utility, safety of potential loops
- After construction: will plan $\Pi$ solve problem instance $x$?
- Longstanding issues of plan reuse
  - Triangle tables, macro operators: linear sequences [Fikes et al., 1972]
Existing Approaches

[Levesque, 2005, Winner and Veloso, 2007, Bonet et al., 2009]

Plan Generation
- Observe working plans
- Extract repetitive patterns and make loops
- Loop utility justified largely by prior experience

Plan Execution
- Apply on any problem instance
- No preconditions; instantiation may traverse the entire state space! (EX: [Levesque, 2005])

Fundamental problems behind these limitations: termination, reachability in plans with loops.
Overview

- Abacus Programs
  - Linear Abacus Programs
  - Abacus Programs with Simple Loops
  - Abacus Programs with Complex Loops
  - Theoretical Results

- Translating Plans to Abacus Programs

- Empirical Results
Abacus Programs

Definition

- $\mathcal{R}$ finite set of registers
- $\mathcal{S}$ finite set of states
- $s_0$ initial state
- $s_h$ halt state
- $\ell \mathcal{S} \rightarrow \mathcal{A}$

Actions ($\mathcal{A}$):

- $\text{Inc}(r, s): r + +; \text{goto } s$
- $\text{Dec}(r, s_1, s_2): \text{if } r = 0 \text{ goto } s_1$
  - else $r --; \text{goto } s_2$

Turing-complete $\implies$ reachability is undecidable
Linear Abacus Programs

Can Efficiently Find

- Cumulative action effects
- Branch Conditions
Linear Abacus Programs

Can Efficiently Find

- Cumulative action effects
- Branch Conditions

Notation

- $\vec{R}$: $\langle R_1, R_2, \ldots, R_k \rangle$
- $\Delta^A_i$: Ai's change vector
- $A(i)$: Index of Ai's register
- $\vec{R}_i$: $R_i$
- $\Delta^{1..m}$: $\Delta^{A1} + \cdots + \Delta^{Am}$
Linear Abacus Programs

Can Efficiently Find

- Cumulative action effects
- Branch Conditions

Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\vec{R}$</td>
<td>$\langle R_1, R_2, \ldots, R_k \rangle$</td>
</tr>
<tr>
<td>$\Delta^A_i$</td>
<td>$A_i$’s change vector</td>
</tr>
<tr>
<td>$A(i)$</td>
<td>Index of $A_i$’s register</td>
</tr>
<tr>
<td>$\vec{R}_i$</td>
<td>$R_i$</td>
</tr>
<tr>
<td>$\Delta^{1\ldots m}$</td>
<td>$\Delta^{A_1} + \cdots + \Delta^{A_m}$</td>
</tr>
</tbody>
</table>
Linear Abacus Programs: Preconditions

Sequence of $n$ actions; $\bar{F} = \text{final register values}$

\[
(\bar{R} + \Delta^{1...i-1})_{A(i)} \circ 0, \quad i = 1 \ldots n
\]

\[
\bar{F} = \bar{R} + \Delta^{1..n}
\]
Abacus Programs with Simple Loops

Abacus Programs Translation Examples

Siddharth Srivastava

Applicability Conditions for Plans with Loops
Abacus Programs with Simple Loops

\[ \text{LoopIneq}(\bar{R}^0) \equiv (\bar{R}^0 + \Delta^1 \ldots \Delta^{i-1})_{A(i)} \odot 0 \quad i = 1 \ldots n \]
Abacus Programs with Simple Loops

- LoopIneq($\bar{R}^0$) $\equiv$ 
  $$(\bar{R}^0 + \Delta^1_{i-1})_{A(i)} \circ 0 \quad i = 1 \ldots n$$

- $\bar{R}^x = \bar{R}^0 + x \cdot \Delta^1_{i=n}$

Siddharth Srivastava
Applicability Conditions for Plans with Loops
Abacus Programs with Simple Loops

- LoopIneq($\bar{R}^0$) \(\equiv\) 
  \((\bar{R}^0 + \Delta_1^{i-1})_{A(i)} \circ 0\) \(i = 1 \ldots n\)

- \(\bar{R}^x = \bar{R}^0 + x \cdot \Delta_1^{n}\)

- For \(l\) complete iterations:
  
  \(\text{LoopIneq}(\bar{R}^0) \land \text{LoopIneq}(\bar{R}^1) \land \cdots \land \text{LoopIneq}(\bar{R}^{l-1})\)

  \(\equiv \text{LoopIneq}(\bar{R}^0) \land \text{LoopIneq}(\bar{R}^{l-1})\)
Abacus Programs with Simple Loops

- \( \text{LoopIneq}(\bar{R}^0) \equiv (\bar{R}^0 + \Delta^{1...i-1})_{A(i)} \circ 0 \quad i = 1 \ldots n \)
- \( \bar{R}^x = \bar{R}^0 + x \cdot \Delta^{1..n} \)
- For \( l \) complete iterations:
  \( \text{LoopIneq}(\bar{R}^0) \land \text{LoopIneq}(\bar{R}^1) \land \cdots \land \text{LoopIneq}(\bar{R}^{l-1}) \equiv \text{LoopIneq}(\bar{R}^0) \land \text{LoopIneq}(\bar{R}^{l-1}) \)
- \( \bar{F} = \bar{R}^l = \bar{R}^0 + l \cdot \Delta^{1...n} \)
Abacus Programs with Simple Loops: Preconditions

Necessary and Sufficient Conditions
For \( l \) complete iterations of a loop with \( n \) actions

\[
\text{LoopIneq}(\bar{R}^0) \land \text{LoopIneq}(\bar{R}^{l-1})
\]

\[
\bar{F} = \bar{R}^l = \bar{R}^0 + l \cdot \Delta^1...n
\]
Recall: Approach

- Translate plans with loops into abacus programs (preserve structure)
- Design algorithms for finding preconditions of classes of abacus programs

Need to represent “sensing” actions.
Non-deterministic Actions

**NSet Action**

\[
NSet(r, s_1, s_2) \quad \text{set } r \text{ to } 0; \text{ goto } s_1 \in S \\
\text{or} \quad \text{set } r \text{ to } 1; \text{ goto } s_2 \in S
\]

Abacus programs don’t need \textit{NSet} for computational power (Turing-complete). \textit{NSet} allows in-place translation of plans with sensing actions.
Non-deterministic Actions

**NSet Action**

\[ NSet(r, s_1, s_2) \]

- set \( r \) to 0; goto \( s_1 \in S \)
- or set \( r \) to 1; goto \( s_2 \in S \)

- Abacus programs don’t need \( NSet \) for computational power (Turing-complete).
- \( NSet \) allows in-place translation of plans with sensing actions.

Find preconditions in terms of effect counts
Simple Loops with Shortcuts
More commonly considered “nested” loops
Simple Loops with Shortcuts

More commonly considered “nested” loops ....can be translated by changing the start node.
Simple Loops with Shortcuts

Siddharth Srivastava
Applicability Conditions for Plans with Loops
Monotonicity

Monotone Shortcuts

The *net* change on a register due to every simple loop created by shortcuts must be in the same direction (positive/negative).
Suppose: \( m \) simple loops; \( k_1, \ldots, k_m \) iterations

\[
\bar{F} = R^0 + \sum_{i=0}^{m} k_i \Delta^{\text{loop}_i}
\]

- \( k_i \) iterations of loop \( i \): no register can fall below zero.

Need: least intermediate value of every register \( \geq 0 \).
Constraining Minimum Register Values

Constraints on Min Values

\( m \) simple loops; \( k_1, \ldots, k_m \) iterations

\[
R_j \in \mathcal{R}^+ : \quad R_j^0 + \delta_j \geq 0
\]

\[
R_j \in \mathcal{R}^- : \quad R_j^0 + \sum_{i=0}^{m} k_i \Delta_{\text{loop}}^i + \delta_j - \Delta_{\text{loop}}^j \geq 0
\]

Notation

\( \mathcal{R}^+ \) registers with net positive change

\( \mathcal{R}^- \) registers with net negative change

\( \hat{j} \) index of loop with greatest partial negative change on \( R_j \).
### Order Dependence

#### Constraints on Minimum Register Values

<table>
<thead>
<tr>
<th>Condition</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_j \in R^+$</td>
<td>$R_j^0 + \delta_j \geq 0$</td>
</tr>
<tr>
<td>$R_j \in R^-$</td>
<td>$R_j^0 + \sum_{i=0}^{m} k_i \Delta^{\text{loop}_i} + \delta_j - \Delta^{\text{loop}_j} \geq 0$</td>
</tr>
</tbody>
</table>

#### Example

- **Loop**
  - **loop$_1$**: increase $R_1$ by 3.
  - **loop$_2$**: decrease $R_1$ by 2, then increase it by 5.

Efficiently expressing general order dependent constraints: open problem!
Order Dependence

Constraints on Minimum Register Values

\[ R_j \in \mathcal{R}^+ : \quad R^0_j + \delta_j \geq 0 \]
\[ R_j \in \mathcal{R}^- : \quad R^0_j + \sum_{i=0}^{m} k_i \Delta^{\text{loop}_i} + \delta_j - \Delta^{\text{loop}_j} \geq 0 \]

Example

- **loop**₁ increase \( R_1 \) by 3.
- **loop**₂ decrease \( R_1 \) by 2, then increase it by 5.

**Constraint** \( R^0_1 - 2 \geq 0 \) (sufficient condition)

Efficiently expressing general order dependent constraints: open problem!
Order Dependence

Constraints on Minimum Register Value

$R_j \in \mathcal{R}^+ : R_j^0 + \delta_j \geq 0$

$R_j \in \mathcal{R}^- : R_j^0 + \sum_{i=0}^{m} k_i \Delta^{\text{loop}_i} + \delta_j - \Delta^{\text{loop}_j} \geq 0$

Example

-loop$_1$ increase $R_1$ by 3.
-loop$_2$ decrease $R_1$ by 2, then increase it by 5.

Constraint $R_1^0 - 2 \geq 0$ (sufficient condition)

Accurate Conditions Order dependent!

Efficiently expressing general order dependent constraints: open problem!
Equality Constraints, Order Dependence

Non-spurious equality constraint: inherently order dependent

Equality constraints, $\delta_j \neq \Delta_{loop_j}$ induce order dependence (details in paper).
Let $S$ be a node in an abacus program $\Pi$, all of whose strongly connected components are simple loops or simple loops with monotone shortcuts; $\bar{F}$: a vector of register values.

**Theorem**

We can compute a disjunction of linear constraints on the initial register values ($\bar{R}^0$) for reaching $S$ with $\bar{F}$. The computed conditions are necessary and sufficient if every simple loop with shortcuts is order independent, and sufficient otherwise.
Translating Plans To Abacus Programs

Choose c: atL1(c) & obj(c) → LoadT(c) → MvToL2T() → UnloadT() → MvToL1T()
Translating Plans To Abacus Programs

Works for all unary, and a class of binary domains
Empirical Results

\[
s_f^3 = m_f^3 = \sum_{i=0}^{7} k_i;
\]

\[
s_f^1 = s_1^0 - \sum_{i=0}^{7} k_i - k_0 - k_5 - k_6 - k_7 = 0
\]
# Empirical Results

## Time Taken to Compute Preconditions

<table>
<thead>
<tr>
<th>Problem</th>
<th>Time (s)</th>
<th>Problem</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>0.01</td>
<td>Prize-A(7)</td>
<td>0.02</td>
</tr>
<tr>
<td>Corner-A</td>
<td>0.00</td>
<td>Recycling</td>
<td>0.02</td>
</tr>
<tr>
<td>Diagonal</td>
<td>0.01</td>
<td>Striped Tower</td>
<td>0.02</td>
</tr>
<tr>
<td>Hall-A</td>
<td>0.01</td>
<td>Transport</td>
<td>0.01</td>
</tr>
<tr>
<td>Prize-A(5)</td>
<td>0.01</td>
<td>Transport (conditional)</td>
<td>0.06</td>
</tr>
</tbody>
</table>
Conclusion

- An approach for computing summarized effects of loops of actions
- Use during construction and for precondition evaluation
- Future work:
  - Greater translation to abacus programs (counts of properties)
  - Further categorization of tractable classes
  - Expression of order dependent constraints
  - Use of symbolic precondition evaluation for non-numeric branches.
Preconditions for a Simple Loop with Shortcuts

\[ I = \sum_{i=0}^{m} k_i; \quad \bar{F} = \bar{R}^0 + \sum_{i=0}^{m} k_i \Delta^{\text{loop}}_i \]

- \( R_j \in \mathcal{R}^- : \forall_{x=0_j,\ldots,m_j} \{ k_{i<x} = 0; k_x \neq 0; \}

- \( R_j^0 + \sum_{i\geq x} k_i \Delta^{\text{loop}}_i + \delta_x - \Delta^{\text{loop}}_x \geq 0 \}

\( R_j \in \mathcal{R}^+ : \forall_{x=0_j,\ldots,m_j} \{ k_{i<x} = 0; k_x \neq 0; R_j^0 + \delta_x \geq 0 \} \]
