ITEM #190 - Feasible Horizontal Bridging Path Trimmer

Conversation： Feasible Path Trimming

20251230

Authors: Sizhe Tan & GPT-Obot

ITEM #190 — Feasible Horizontal Bridging Path Trimmer

Y-Anchored Reverse Pruning for ACLM Horizontal Bridging

---

1. Motivation

In ACLM Horizontal Gap Bridging, it is often straightforward to construct a **Feasible Horizontal Bridging Path** between an extended start set **X′** and an extended target set **Y′**. Such a path is *execution-feasible* but frequently contains **redundant states, branches, and calling links** that do not contribute to the actual target **Y**.

For engineering deployment, explanation, testing, and code generation, we require an **Exact Path**:

a minimal, goal-sufficient calling path from **X** to **Y**,
derived from the feasible result without re-solving the bridging problem.

This item introduces a **simple, deterministic, and engineering-friendly trimming algorithm** that converts a *Feasible Horizontal Bridging Path* into an *Exact Path* by **Y-anchored reverse pruning**.

---

2. Problem Definition

Given:

- A feasible horizontal bridging subgraph
  `G = (V, E)`
  obtained from ACLM Horizontal Bridging
- Extended start set `X'`, where `X ⊆ X'`
- Extended target set `Y'`, where `Y ⊆ Y'`
- Target requirement set `Y` (the real goal)

Goal:

Derive a trimmed subgraph `G*` (or canonical path) such that:

- `G*` contains only states and calling links that **contribute to reaching Y**
- All redundant branches related only to `Y' \ Y` are removed
- The result is an **Exact Path from X to Y**, suitable for code extraction

---

3. Core Idea

A node or calling link is **useful if and only if** it lies on at least one path that **can reach Y**.

Therefore:

Instead of pruning forward from X,
we **anchor at Y and prune backward**,
removing everything that cannot contribute to Y.

This turns feasible-but-loose structure into a **goal-minimal executable structure**.

---

4. Trimming Algorithm

Step A — Remove Unused Target States

1. Identify unused terminal states:

1. $\texttt{Dead}_0$ `= Y' \ Y`
2.
3. For each `v ∈` $\texttt{Dead}_0$, delete calling links that feed into `v`.

This immediately cuts branches that serve only unused targets.

Step B — Collect Newly Unused Predecessor States

After Step A, some predecessor states may no longer lead to any valid target.

- If a state has **no outgoing path that can eventually reach Y**,
  it becomes unused.

Collect such states and mark them as dead.

Step C — Iterative Backward Pruning

Repeat Step B:

- Each time a state is removed,
  its predecessors may become unreachable from Y
- Continue until no new unused states appear

This process always converges, since the graph is finite.

Step D — Obtain the Exact Path (X → Y)

After pruning stabilizes:

- The remaining subgraph is the **Exact Horizontal Bridging Subgraph**
- Optionally:
  - Restrict further to nodes reachable from `X`
  - Extract a canonical path (e.g., minimal cost or minimal steps)

The resulting structure represents **exactly the code required** to bridge from X to Y.

---

5. Equivalent Engineering Formulation (Recommended)

Instead of iterative deletion, the same result can be obtained more robustly:

1. Initialize `Live = Y`
2. Traverse the graph **backward** (reverse edges):
   - Mark all states that can reach `Y`

3. Keep only marked states and their calling links

This reverse-reachability formulation is:

- Order-independent
- Cycle-safe
- Linear in graph size
- Easy to implement and test

---

6. Correctness Intuition

- Any state not retained **cannot reach Y**
  → removing it cannot affect Y
- Any retained state **contributes to at least one Y-reaching path**
  → removing it would break feasibility

Therefore, the algorithm produces the **minimal Y-sufficient subgraph** within the feasible result.

---

7. Engineering Properties

- **Deterministic**: no heuristic scoring required
- **Linear complexity**: `O(|V| + |E|)`
- **Cycle-safe**: handles recursive and mutual calls
- **Composable**:
  - Can be applied after any feasible bridging algorithm
  - Independent of how feasibility was obtained
- **Explainable**:
  - Each retained state has a provable contribution chain to Y

---

8. Role in ACLM Architecture

This trimmer acts as a **mandatory post-processing stage**:

```
Horizontal Bridging (Feasible)
        ↓
Feasible Subgraph (X′ → Y′)
        ↓
ITEM #190 Trimmer
        ↓
Exact Path / Exact Code (X → Y)
```

It cleanly separates:

- *Search feasibility*
  from
- *Execution minimality*

---

9. Summary

ITEM #190 formalizes a key insight:

Feasibility first, exactness later —
and exactness is achieved by Y-anchored reverse pruning.

This mechanism is simple, reliable, and essential for making ACLM Horizontal Bridging **engineering-grade, explainable, and deployable**.

ITEM #190 — 可行水平桥接路径裁剪器（中文版）

以 Y 为锚点的反向裁剪算法

---

1. 动机

在 ACLM 的 **Horizontal Gap Bridging** 中，
构造一条从 **X′ 到 Y′** 的**可行桥接路径**通常并不困难。

但该路径往往：

- 包含多余状态

- 带有仅服务于非目标终点的分支
- 不适合直接生成或解释最终代码

工程上真正需要的是：

一条 从 X 到 Y 的最小必要调用路径

而不是重新解一次桥接问题。

---

2. 问题定义

已知：

- 一条可行水平桥接子图 `G(V, E)`
- 扩展起点集 `X′`，真实起点 `X ⊆ X′`
- 扩展终点集 `Y′`，真实目标 `Y ⊆ Y′`

目标：

从 `G` 中裁剪出：

- 仅保留**对 Y 有贡献**的状态与调用边
- 删除所有与 `Y′ \ Y` 相关的冗余结构
- 得到 **X → Y 的精确可执行路径**

---

3. 核心思想

一个状态是否应该保留，只有一个标准：

它是否位于某条最终能够到达 Y 的路径上

因此：

- 不从 X 正向猜测
- 而是**从 Y 反向裁剪**
- 删除所有“永远到不了 Y”的状态

---

4. 裁剪算法

步骤 A — 删除无用终点

- 定义：

- `Dead₀ = Y′ \ Y`
-
- 删除所有指向 `Dead₀` 中状态的调用链接

步骤 B — 收集新产生的无用前驱状态

- 若某状态在当前图中：
  - 不再能通过任何路径到达 Y
    → 则该状态变为无用

步骤 C — 迭代反向裁剪

- 删除新发现的无用状态
- 重复，直到不再产生新的无用节点

该过程必然收敛。

步骤 D — 得到 X → Y 的精确路径

- 剩余子图即为 **Exact Horizontal Bridging Subgraph**
- 可进一步：
    - 限制为 X 可达部分
    - 提取一条规范化路径作为最终代码

---

5. 推荐工程实现形式

更稳健的等价实现：

1. 从 Y 出发
2. 沿调用边**反向遍历**
3. 标记所有可到达 Y 的状态
4. 删除其余状态与边

这是最推荐的工程实现方式。

---

6. 正确性直觉

- 被删除的状态 **永远无法到达 Y**
- 被保留的状态 **至少存在一条贡献到 Y 的链路**

因此，裁剪结果是：

在可行解空间内，对 Y 最小且充分的结构

---

7. 工程特性

- 线性复杂度
- 对环安全
- 不依赖启发式
- 强可解释性
- 可作为 ACLM 的标准后处理阶段

---

8. 在 ACLM 中的位置

```
可行水平桥接
        ↓
可行子图（X′ → Y′）
        ↓
ITEM #190 裁剪器
        ↓
精确路径 / 精确代码（X → Y）
```

---

9. 总结

ITEM #190 确立了一条重要工程原则：

先解决“能不能”，
再用以 Y 为锚点的反向裁剪解决“要不要”。

这使 ACLM 的水平桥接真正具备了
工程可落地性、可解释性与可维护性。
