**Sudoku Solver Technical Report**

게임소프트웨어전공 B977018 배선우

1. **Project Goal**

스도쿠 문제를 풀기 위한 Complete하고 Optimal한 PDDL을 만드는 것이 목표입니다.

1. **Approach**

우선 스도쿠 문제를 테이블화 시켰습니다.

텍스트, 번호, 달력, 낱말맞추기 퍼즐이(가) 표시된 사진

자동 생성된 설명

가로 라인을 알파벳 라인으로 linea부터 linei까지 정의하고,

세로 라인을 숫자 라인으로 line1부터 line9까지 정의했습니다.

이렇게 나눈 라인을 problem의 object에 정의했습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

또한 숫자들을 정의하고, 1~9까지의 숫자가 들어갈 작은 사각형들 또한 정의했습니다.

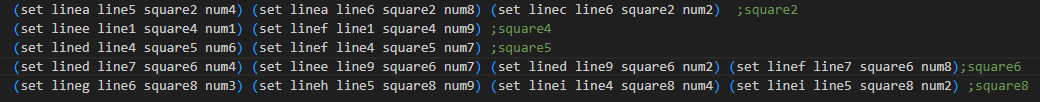
**Domain**

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**Predicates**

1. (has ?x ?y)는 라인 또는 사각형이 숫자 ?y를 가지고 있는 상태를 의미합니다. 이를 통해 이미 ?y가 ?x의 특정 타일에 존재할 경우, ?x가 가지고 있는 다른 타일에 ?y가중복으로 작성되는 것을 방지합니다.
2. (written ?x ?y)는 가로 라인인 lineh와 세로 라인인 linev가 교차하는 지점(타일)에 숫자가 존재하는 것을 의미합니다. 처음에는 81개의 타일을 모두 정의해서 (written ?x)의 형태로 사용하려고 했었는데, 최적화를 시키면서 타일을 라인 2개가 교차하는 것으로 정의할 수 있어서 이러한 형태를 사용했습니다. 이를 통해 이미 해당 타일에 숫자가 존재하는 경우, 다른 숫자로 덮어 씌워지는 것을 방지합니다.
3. (passing ?x ?y)은 ?x라는 라인이 ?y라는 사각형을 지나가는 것을 의미합니다.   
   이는 가로 라인과 세로 라인 두개를 같은 ?y로 지정하여 2개의 라인이 교차하는 지점의 타일이 ?y의 사각형에 속하는 것을 의미합니다.
4. (horizontal ?x)와 (vertical ?x)는 ?x라는 라인이 각각 가로 라인인지, 세로 라인인지를 구분하는 역할을 합니다. 이것으로 이후 설명할 write-number 액션에서 똑같은 유형의 라인이 지정되는 것을 방지합니다.
5. (set ?x ?y ?z ?i)는 미리 배치되어 있는 타일을 지정하는 역할을 합니다.



Problem의 init에서 정의하여 사용합니다.

1. (setcomplete ?x ?y)는 ?x, ?y 라인이 교차하는 지점에 배치가 완료되었음을 의미합니다. 따라서 set을 건너뛰고 write를 하게 되는 것을 방지합니다.

스크린샷, 텍스트, 폰트, 라인이(가) 표시된 사진

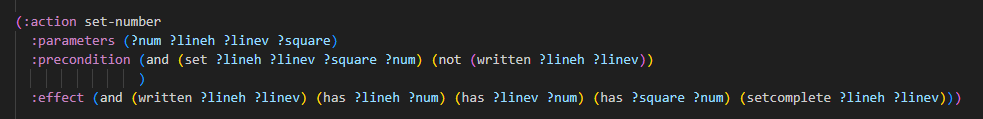
자동 생성된 설명

Problem의 goal에 정의하여 사용합니다.

1. (writtennum ?x ?y ?z)는 ?x, ?y가 교차하는 tile에 ?z라는 숫자가 있음을 의미합니다. 이는 이미 작성한 숫자를 지울 때, 특정 타일에 무슨 숫자가 있는지 확인할 때 사용됩니다.

**Actions**

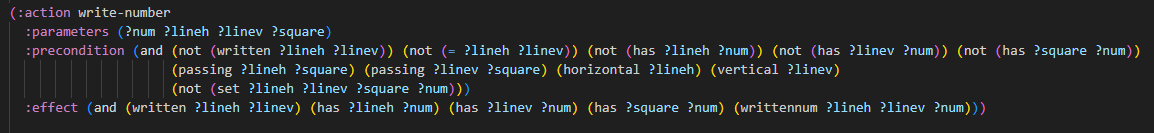
1. **Set-number**



Predicate의 (set ?x ?y ?z ?i)를 이용한 action입니다. Precondtion에 (set ?lineh ?linev ?square ?num)을 똑같이 정의하고, (not (written ?lineh ?linev))를 정의하여 set-number를 다른 action보다 먼저 사용되게 합니다.

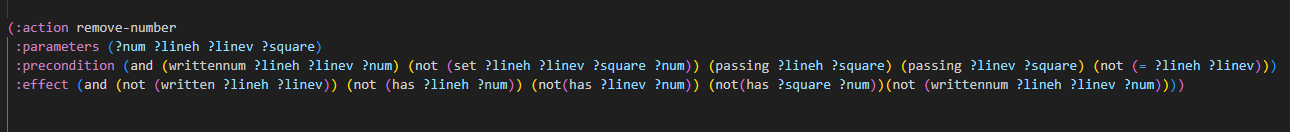
effect로 (written ?lineh ?linev)를 정의하여 해당 타일에 중복으로 숫자가 배치되는 것을 막고, 각각의 라인과 사각형에 (has ?x ?num)을 적용하여 ?num이 중복으로 적용되는 것을 막습니다. 마지막으로 (setcomplete ?lineh ?linev)를 통해 goal에 setcomplete를 정의하여 set-number가 반드시 사용되도록 합니다.

1. **Write-number**



Set-number와 거의 유사합니다. 하지만 precondition에서 몇 가지 조건이 더 정의되어 있습니다. 같은 타일에 연속으로 적용되는 것을 막고, 같은 라인과 사각형에 중복된 숫자가 없어야 하며, 두 개의 라인이 각각 horizontal 라인이거나 vertical 라인이여야 합니다. 또한 passing을 통해 각각의 라인이 교차한 타일이 어떤 사각형에 속해 있는 지 또한 확인할 수 있습니다. 이를 통해 결과적으로 숫자가 타일에 적용되었을 때, 각각의 라인이 그 숫자를 has하게 되고 사각형 또한 숫자를 가지게 됩니다. 마지막으로 writtennum을 통해 해당 타일이 어떤 숫자를 가지고 있는지 알 수 있게 됩니다.

1. **Remove-number**

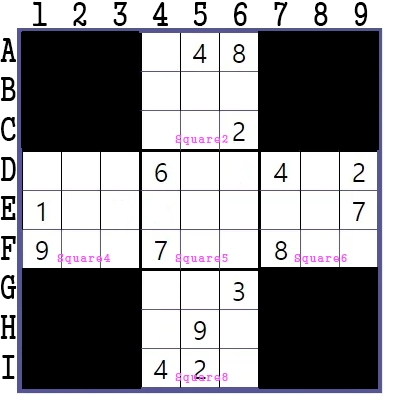


Write-number를 통해 배치한 숫자를 제거할 때 사용합니다. 이는 골에 도달하기 위해 재배치 해야할 숫자가 있을 때 사용됩니다.

**Problem**

Problem에서 여러가지 간소화를 거쳤습니다.

우선적으로 9x9의 스도쿠 테이블을 사용하려고 했으나, 지속적으로 타임아웃이 발생해 스도쿠 테이블을 +모양으로 줄이고, 미리 배치된 숫자 또한 개수를 줄였습니다. 이러한 문제가 발생한 이유는 이후 서술하겠습니다.



테이블이 간소화되었기 때문에, Problem 또한 그에 맞춰 간소화되었습니다.

스크린샷, 텍스트, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

사용하지 않는 square를 줄이고, 그로 인해 init에 정의할 (passing ?x ?y)가 절반 가까이 줄어들었으며 goal또한 많이 축소되었습니다.

테이블을 축소화 했기 때문에, 가능한 경우의 수가 많이 줄었기 때문에 타임아웃이 발생하지 않았습니다.

1. **Result**

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명 스크린샷, 다채로움, 그래픽, 라인이(가) 표시된 사진

자동 생성된 설명

플랜을 Run했을 때 나온 Output과 그것을 기반으로 작성한 스도쿠 테이블입니다.

각각의 라인과 작은 사각형에 1부터 9까지 숫자가 있어야 한다는 규칙이 잘 지켜졌습니다. 또한 흥미로운 점은 hill climbing 알고리즘을 이용하는 플래너를 사용했기 때문에, set-number이후 write-number를 할 때 숫자 1부터 차례대로 타일에 작성하게 되었습니다.

추측이지만 이것이 이후 설명할 타임아웃의 원인이 됐을 것 같습니다.

1. **Discussion**

결과적으로 Complete하지만 Optimal하지 않은 PDDL이 완성되었습니다. 타임아웃의 경우를 제외하면 다른 스도쿠 테이블에서도 항상 답을 도출했습니다. Formal 하려면 remove-number를 사용하지 않고, 항상 한 번에 올바른 답을 작성해야 합니다. 따라서 스도쿠 문제를 푸는 알고리즘이 Formal하기는 어려울 것 같습니다.

타임아웃은 Hill Climbing 알고리즘을 사용한 플래너로 플랜을 도출하려고 했기 때문에, 순서대로 작업을 수행하면서 점점 전진하는 형태로 사용하게 됩니다. 하지만 주로 스도쿠 문제를 해결하는데 사용되는 알고리즘은 DFS의 일종인 Backtracking 알고리즘으로, 문제 해결을 위해 전진했다가 해결되지 않을 경우 성공한 지점까지 돌아와서 다른 경우의 수를 향해 다시 전진하는 방식의 알고리즘입니다.

따라서 이번에 사용한 Hill Climbing 알고리즘은 스도쿠 테이블이 커지고 경우의 수가 많아지면, 특정 숫자를 지우고 다른 숫자를 작성할 경우에 이미 작성되어 있는 다른 숫자에도 큰 영향을 주게 됩니다. 따라서 지속적으로 연쇄 작용이 발생하여 러닝타임이 길어지는 것 같습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이를 더욱 명확하게 확인할 수 있는 부분이 이 부분입니다. 전체적으로 1부터 9까지의 가능한 숫자 작성을 마치고 이후 수정 작업이 들어가기 때문에, 답이 단 하나로 정해져 있는 어려운 난이도의 스도쿠 문제는 수정 시간이 매우 길어질 것으로 추측됩니다.

결론적으로 이번에 사용한 Hill Climbing 알고리즘은 스도쿠 문제를 해결하는 것은 가능하지만, 효율면에서 적합하지 않습니다.

1. **Future Work**

우선 타임아웃 설정이 10분으로 고정되어 있는 것을 해제하고 좀 더 시간을 늘려서 완전한 9 x 9 크기의 스도쿠 문제를 해결해보고 싶습니다. 또한 시간을 단축하기 위해서 숫자가 이미 확정된 타일을 분석하고 이것을 휴리스틱 함수로 사용해 다른 경우의 수가 적은 숫자부터 미리 작성하는 방식으로 한다면 러닝타임을 더욱 줄일 수 있을 것 같습니다.

A\* 알고리즘을 사용하여 플랜을 도출한다고 가정했을 때, 휴리스틱 함수는 해당 타일에 배치가능한 숫자의 가짓수로 하여 숫자가 적으면 적을수록 먼저 탐색하도록 하면 현재 Hill Climbing 알고리즘에서 발생하는 많은 시행착오를 줄일 수 있을 것으로 예상됩니다. 초반에 잘못 배치하게 되는 숫자가 적어지므로 러닝 타임 또한 현재의 경우보다 많이 짧아질 것 같고, 현재로서는 타임아웃으로 인해 불가능한 9 x 9 테이블의 문제를 해결할 수 있게 될 것 같습니다.

9 x 9 크기의 스도쿠 문제를 풀 수 있게 된다면, 인공지능이 plan을 도출하는 시간과 제가 직접 푸는 것의 시간을 비교하여 누가 더 빨리 풀지 비교해 보고 싶습니다.

1. **Appendix**

대부분의 플래너들이 리눅스 환경에서 지원하는 것이 많았습니다. 윈도우 환경에서도 다양한 탐색 알고리즘을 지원하는 플래너가 있었다면 더 좋은 결과를 볼 수 있었을 것 같습니다.

인터넷 서치나 ChatGPT 등의 외부 참조를 하지 않고 제작했습니다.