FLYING LOGIC

Flying Logic で思考する

ロバート・マクナリー

Version 1.0.2

日本語訳

2014 年 7 月 11 日

Sciral

Sciral

726 E. Colorado Ave. #9

Glendora, CA 91740

FlyingLogic.com

翻訳者まえがき

日本語訳について

本書は、TOC 思考プロセスのわかりやすい入門書として評判の高い Thinking with Flying Logic の日本語訳です。翻訳については、Flying Logic の開発元の了解を得ています。

翻訳についてお気づきの点がありましたら、こちらまでご連絡ください。内容について確認させていただくことがありますので、お名前とメールアドレスを必ず記載していただけるようお願いいたします。

中津山　恒
http://nakatsuyama.biz

謝辞

日本語訳の作成について快諾いただいた、開発元の Joe Pearce 氏、Robert McNally 氏に感謝します。また、非常にご多忙な中を縫って日本語訳を詳細にチェックしていただいた、羽生田栄一氏（株式会社 豆蔵 取締役 CTO）に心より謝意を表します。羽生田氏には重要な訳語の修正案もご提案いただき、それをもとに分かりやすさを優先した訳語を当て、必要に応じて訳注を補足しました。

第 1 部　はじめに

本書について

Flying Logic は、改善を支援するソフトウェアである。本書「Flying Logic で思考する」（Thinking with Flying Logic）は、Flying Logic が支援する中心的な技法を紹介する。読者が Flying Logic を使っていない場合でも、仕事と私生活を改善するパワフルな方法の有用な入門書だと評価していただけることを願っている。

「Flying Logic で思考する」は、次の 2 つの文書と組である。「ようこそ Flying Logic へ（Welcome to Flying Logic）」は Flying Logic の意義を説明し、「Flying Logic ユーザーズガイド（Flying Logic User's Guide）」はソフトウェアの詳細な操作方法を説明している。旅行にたとえると、「ようこそ Flying Logic へ」が読者に旅行への興味を抱かせ、「Flying Logic ユーザーズガイド」が読者に自動車の運転を教え、「Flying Logic で思考する」が道路地図として読者の望む場所へ案内する。

しかし、「Flying Logic で思考する」は、それが扱う技法の網羅的な入門書ではない。実際、表面をなぞっているだけである。とくに、Flying Logic を開発するきっかけとなった TOC（制約の理論）と TOC 思考プロセスには、数多くの文献、書籍、論文、Web サイト、講義、会議、コンサルタント、講師、学会、実装者、研究、および成功事例が存在している。著者は Flying Logic がこのパズルに不可欠なピースだと信じており、読者すべてに、こうした他の優れたリソースも探求するよう強く勧めたい。そのうちいくつかは、付録に掲載した。

優れた思考への鍵

本書の大半は、各技法を実践するためのステップごとの説明に費やしている。しかし、この序章では、著者が見つけた、効果的な思考とコミュニケーションを行うための習慣を作る、いくつかのアイディア、態度、振る舞いについて簡単に説明したい。これらは、Flying Logic を効果的に使うための究極の鍵である。

論理と感情

「論理」は一般に、冷たく複雑な話題だと見られている。高等数学であり、お堅い教

授や、SF のコンピュータ、感情のないエイリアンのイメージを連想させる。しかし実際には、**私たちは皆考えている**ので、多少なりとも技能をもって論理を用いている。

論理が単に**思考のルール**だということは、あまり理解されていない。危険だが道路上のルールを知らないでも車を運転できるのと同様に、論理のルールを理解しないでも思考することはできる。これらのルールは極めて強力で、しかも幸運なことに非常に単純である。しかし残念なことに、子供のときに、読み書きを学ぶのと同じくらい自然に論理のルールを教わることはまずない。人間を感情のない機械へと変えることは決してなく、私たち人間は生来、感情的な心と論理的な思考を組み合わせたときに、最も幸福で生産的である。

与えられた話題を「どう感じるかを分かっている」として「論理的になること」に抵抗する人もいる。しかし、強烈な感情や本能を感じたときには、その感情の裏には常に原因があると認識することが重要である。もし結果として生じた感情だけを認め、原因を深く理解しようとしないなら、思考の理性と感情を断絶させてしまう。この断絶は**認知的不協和**を引き起こす。これは、対立することを信じようとしたり、対立する方法で振る舞おうとしたりするときに生じるストレスである。認知的不協和は諸刃の剣である。意見をよりよく変えよう（すなわち、現実をより反映する）という動機を与える一方、現実を反映しないと感じる方法を**正当化**する方向へと導くこともある。どちらの行動も短期的には認知的不協和の不快感を和らげるが、現実との一致から遠く外れていくことから、正当化によって結局より深い問題に入り込んでしまう。

感情だけで行動しようとすることには、もうひとつ欠点がある。こうした行動は、理性的な思考が予期し避けられるかもしれない、意図しなかった結果を招きやすい。もちろん、理性は別の働きもする。「純粋に理性的」であろうとし、**理由を軽んじる**ことで強い感情を無視したら、やはり不協和を起こしてしまう。

解決策は、感情や本能の裏にある原因（または理由）を表面化させる習慣を身につけることである。そうすることで、感情を検証し、それを効果的な計画に取り入れることができる。

ありがたいことに思考は実践で**習得できる技能**であり、そうすることによって感情の価値を損ねるのではなく、むしろ**補う**のである。

コミュニケーションと批判

重大なことを独力で成し遂げられることは滅多にない。他者のさまざまな貢献を必要とする。誰も孤島ではないので、他者と効果的にコミュニケートしなければならない。他者のニーズを理解したり、経験と知見を役立てたり、協力を求めて交渉したりするために。

近すぎて状況を理解できないことも多い。状況の詳細に引き込まれて「木を見て森を見ない」のだ。状況をよく理解していると考えているときや、すべての選択肢と正解を知っていると思っているときがそうだ。この場合は、計画の評価と批判のために他者を引き入れることに価値があるだろう。そうすることで、心に「光と空気」を取り入れ、新鮮さが失われて生産的でなくなった思考方法から脱け出す助けになる。

ゴッドファーザー Part II で、マイケル・コルレオーネは「友人を近くに置け。しかし敵はもっと近くに置け」と言っている。皮肉なことに、実りのある批判は、積極的に反対する人から得られることが多い。エイブラハム・リンカーンは、おそらく最も偉大なアメリカ合衆国大統領だが、彼の政策に激しく反対する人物の中から重要閣僚を選んでいたことで有名である。最終的に批判に同意しようがしまいが、批判から多くのことを学べることがある。重要なことは、学んだときに、ものの見方が変わるのを受け入れることである。

議論と名誉

議論というと、しかめ面、怒りのジェスチャー、怒声を思い起こす人が多いだろう。些細な悪口、許されない不満の数々、他の感情的なパワープレイを思い浮かべる。最も注意すべきは、**自分たちが思う通りにする**ために議論すると考えてしまうことだ。自分たちが**正しい**、他人は**間違っている**、と。しかし、このようなやり取りは議論ではない。それは**争い**だ。争いには勝者があるかもしれないが、敗者も確実にあり、全員が傷つく。

本物の議論とは、真実を一緒に追究することだ。公正な議論では、人々は熱情的であることもあるが、運転者が交通法規を守るのと同じように、論理の規則に従う。人々は様々な観点から予断をもって状況を理解するが、取るべき立場は、**Win/Win を目指している**提案と捉えられるようにすることだ。たとえ、初めの段階では程遠いにしても。実際、「君が私のやり方を学んだらすぐ、我々はうまくやっていけるだろう」という断固とした言い方でさえ、「折り合いをつける」という共通の目的を示している。

真実の探求と捉えられる議論では、新しい情報やアイディアに対して立場を変化させることは弱さや優柔不断とは見られず、強く、成熟した、尊敬すべき人物のみが受け入れられる試練と見られる。より実践的には、誰もが単に**自分の思うようにする**だけでなく、Win/Win の解という形の**よりよい結果を得よう**と望むことだ。

統制と影響

変化を起こす方法を検討しているときは、自分が円の中心に立っていると考えるとよい。手が届いて直接触れられるものが、自分の**統制範囲**である。起こしたい変化すべてが統制範囲内にあるなら、単純に事を進めて変化を起こすことができる。

しかし、物事はそう単純でないのが普通である。私たちの心象では、統制できるのは腕が届く範囲にすぎない。統制範囲は、いつも非常に狭い。しかし、統制範囲を超えたところから**影響圏**が始まる。これらに直接手を触れることはできないかもしれないが、他者の協力によって変化を引き起こすことはまだ可能である。たとえば、ある企業は自社の製造プロセスを**統制**できるかもしれないが、サプライヤーや顧客には**影響**を及ぼせる可能性があるだけだ。

ものが離れているほど、私たちの影響力は小さくなる。実質的に影響力をもたなくなるところまで。この点が、影響圏の終端である。

影響圏は統制範囲よりもずっと広く、想像よりもおそらく広い。嬉しいことに、影響圏内でポジティブな変化を起こすことは、影響圏を拡大させるという望ましい効果がある。

最適化と部分最適化

統制範囲内の改善に報酬を払うとしたら、自然な反応は何だろうか。この例は、管理職の業績評価を彼らの部門内の効率だけで行うことかもしれない。自然な反応はもちろん、できるだけ自分たちの**統制範囲を狭くする**ことだ。システムの他の部分との明確な境界を定義し、特定の要素（部、課、居室など）の中での効率だけに集中する。この行動は**部分最適化**を招く。システムの一部を最大化または微調整し、そうすることがシステム全体に与える（しばしば有害な）効果を考えることがない。

他方、**影響圏**すべてにおける改善に対して報酬を払うとしたら何が起きるだろうか。この場合、人々の望みは影響圏をできるだけ押し広げることになる。前述のように、影響圏での行動には他者との調整と協調が必要であり、それが全体としてのシステムの気

づきを促す。最終的な結果は**最適化**であり、人々はシステムの目的を達成するために組織的に働く。

最適化は、**プロセス改善**の目的に適用した**システム思考**（システムを部分の集合と見るのではなく、統合されたひとつの総体として見る）の成果である。

ツールと期待値

現実を正確に認識し、知識を体系化し、論理的に考え、計画を立案し、効果的に伝達するために役立つツールが数多く発明された。こうしたツールがあっても、まだ困難な思考を行わなければならず、計画を実行することもまた困難である。新しいツール（たとえば Flying Logic）が導入されると、それらは省力化装置として売り込まれることがよくある。しかし、自動車、電話、コンピュータがあっても仕事が本当に減っているのだろうか。まず間違いなく、変化が加速している世界では、**より多く**の仕事をする。したがって、新しいツールの効果は**労働の削減**ではなく**期待値の増大**だという、実践的な理解が重要である。

スプレッドシートは財務計画に有益だが会計士がなくならなかったように、筆者の希望は、システム思考を行う人々と、現実およびそのシステムをよりよくしていこうという情熱をもった人々を、Flying Logic が大いに助けることである。さらに、Flying Logic が、この活力ある話題により多くの人が加わる一助となることを希望する。

―ロバート・マクナリー 2007 年

第 2 部　TOC 思考プロセス

TOC の概要

ザ･ゴール

TOC（Theory of Constraints＝制約の理論）は、エリヤフ・M・ゴールドラット（エリ）が構築した経営全般の理論であり、ベストセラーとなった彼のビジネス小説「ザ・ゴール」で初めて一般の知るところとなった。彼は、実世界の**システム**全体が、個人、個人間、組織のいずれのものであるにせよ、本来の目的すなわち**ゴール**をもつというアイディアから考え始めた。システムがその目的を達成する能率は、**スループット**と呼ばれる。

制約

スループットのアイディアから、システムが少なくとも 1 つの**制約**をもつことも容易に分かる。制約はシステムのスループットを制限するもので、鎖の最も弱い輪に例えられる。システムがまったく制約をもたないなら、それは無限のスループットをもつ。無限のスループットは不可能だが、システムの主要な制約を注意深く特定して管理すれば、驚くほどスループットを増加可能で**ある**。TOC の目的は、望みうる最も効果的な方法で、制約の管理に必要なツールを個人と組織に提供することである。

初めは製造ラインに適用されたが、TOC の原理は、サプライチェーン、財務、プロジェクト管理、ヘルスケア、軍事計画、ソフトウェアエンジニアリング、戦略まで、幅広い領域に適用されてきた。

TOC は、3 つより多い制約をもつ実世界のシステムは極めて稀であり、実際には唯一の制約が鍵であることが普通だと主張している。直観に反するが、これはシステムが**複雑になるほど**、その部分間の関係が**多くなる**からである。これにより、全体の自由度が下がる結果となる。

このことの重要な示唆は、焦点を当てるべき少数の具体的で影響の大きい範囲を経営者に示すことによって、複雑なシステムや組織の管理を単純かつ効果的にできるということである。主要な制約のパフォーマンスを最大化する、あるいは制約を**緩和する**（制約を弱める）のである。TOC は初め製造作業に適用され、そこでの制約は、製造ラインのボトルネックとなっている特定の機械や処理といった**物理的な制約**が普通であった。この種の制約は特定するのがかなり容易である。しかし、これらの制約が**取り払わ**

れている（すなわち、もう制約ではないところまで緩和されている）という現実の状況では、制約は**方針制約**という別な姿で見つかることがある。これは最終的にシステムのスループットを制限する「いつもやってきた方法」であり、ある種の**部分最適化**に起因することが普通である。つまり、全体の便益に注意を払うことなく、システムの一部を調整することである。方針制約を特定することは難しく、単純な機械や物理的なプロセスよりも管理が難しい。これを行うためだけに、より強力なツールが考案された。

5 つの重要ステップ

あらゆる種類の制約を特定し、管理するために、TOC の開発者たちは「**5 つの重要ステップ**」[1]を定めた。ここでは、継続的なプロセス改善について述べている。（明確化のために、後にステップ 0 が追加された。）

0. システムの目的を**明確化する**。
   システムの成功をどう測るか。
1. 制約を**特定**する。
   システムの目的達成を妨げている資源は何か。
2. 制約を最大限に**活用する**。
   制約している資源の稼働率をできるだけ上げ、システム全体に対して最大の価値を生むことだけをさせるにはどうすればよいか。
3. ステップ 2 で行った決定に、他のすべてのプロセスを**従わせる**。
   制約している資源に必要なものすべてを提供するために、どうすれば全プロセスを整合させられるか。
4. 制約を**緩める**。
   制約している資源をより効果的に管理したとしても必要な改善ができないなら、より多くの資源を確保するにはどうすればよいか。
5. **慣性を避ける**。
   前のステップの結果として、制約は他の資源に移動したか。そうであれば、慣性自身が制約とならないようにする。ステップ 1 に戻る。

5 つの重要ステップを数回繰り返した後、システムのスループットに関する制約がシステムの外に完全に出て、システムの環境へと移動することがある。この例として、製造業者の能力が製品需要よりも大きい状況が挙げられる。この場合もまだ改善が可能だ

[1] 訳注：元の用語は Five Focusing Steps で、「５つの集中ステップ」という意味合いである。

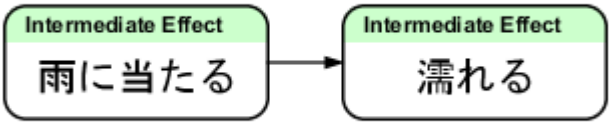
が、そうするには、顧客、経済、そして当初はシステムの環境の状況にすぎなかった他の要素を含むよう、「システム」の概念を拡張する必要がある。

思考プロセス

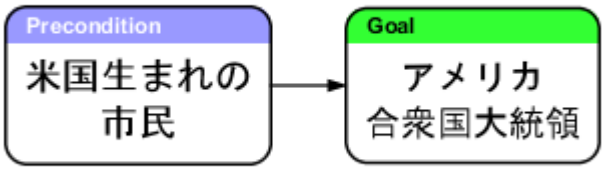
思考プロセスは、TOC の実践者が、中核の制約と、その管理あるいは緩和方法を特定する必要のある組織と働く中で生み出された。これに必要なのは、嘘のように簡単な 3 つの質問だけである：

- **何を**変えるか
- **何に**変えるか
- 変化を**どう起こす**か

思考プロセスは科学的な方法に基づいており、単純な視覚的言語である**思考プロセス図**が導入されている。これは**因果関係**の言語を使って、状況、議論、計画を、記述及び推論するために用いられる。推論には、2 つの基本的な種類がある。**十分原因**と**必要条件**である。



結果に対する十分原因



結果に対する必要条件

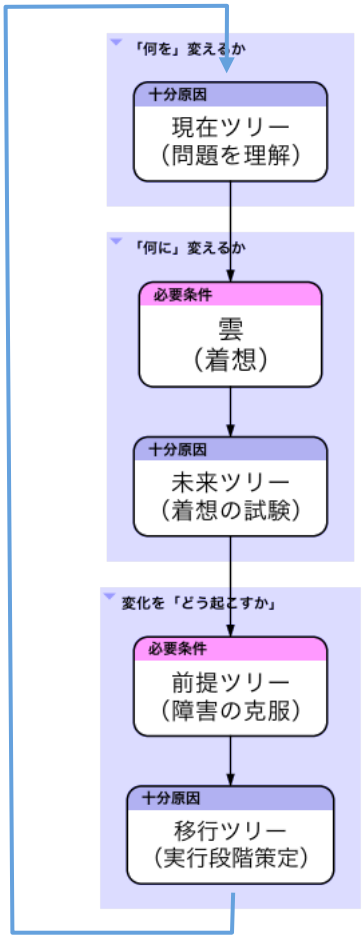
思考プロセスツール

この 3 つの質問に答えるために、**思考プロセスツール**と呼ばれる技法が基本的な思考プロセスから考案された。このツールは、システムの中核的な制約を完全に描き、それ

らを管理する能力を提供する。

ツール	思考プロセス	出発点	結果
現在ツリー（CRT）	十分原因	好ましくない症状	兆候（制約）の中核原因
雲	必要条件	制約の原因となっている認識された対立	可能な Win/Win 解決策
未来ツリー（FRT）	十分原因	提案された解決策	解決策を実現し、新たな問題を避けるために必要な変化
前提ツリー（PRT）	必要条件	主な目的と克服すべき障害	全障害を克服するためのマイルストーン
移行ツリー（TRT）	十分原因	目標	目標を達成するために必要な詳細な行動
戦略＆戦術ツリー（S&T）	必要条件	システムの上位目標	多階層の実現ステップ

最後のツール「**戦略＆戦術ツリー**」は、大規模な変化を短期間で行う必要がある大きな組織で用いられる。しかし、他の 5 つのツールは、個人から家庭や大小企業に至るまで、どのような規模のシステムにも適用可能である。実際の道具箱のように、仕事に適した個々のツールを選択して使うこともできる。あるいは、大部分またはすべてのツールを必要とするかもしれない大きなプロジェクトを実行することも可能である。すべてのツールが使われる場合には、あるツールの「完成品」は次のツールに対する「原材料」として容易に用いることができる。改善は継続的なプロセスなので、5 つの重要ステップを実行するたび、各ツールを繰り返し使うことができる。



成功の計測

改善パズルの最後のピースはフィードバックである。変化の実現にあたっては、改善を計測する明確な方法を導入する必要がある。従来の事業に対して、ゴールドラット博士は、システムのゴールに関する概念を覆すことから始めて 3 つの新たな尺度を生み出した：**スループット**（T）、**在庫**（I）、**業務費用**（OE）である。これらを詳細に議論することは本書のスコープ外だが、これらの尺度と、他の多くの事柄に適用されてきた方法の議論については TOC 知識体系（付録）を参照してほしい。

妥当性基準（CLR）

誰しも自分のアイディアや計画が合理的であって欲しい。しかし、どうやったら合理的だと分かるのだろうか。そもそも合理的であるとはどういう意味なのか。**思考プロセスツール**を用いるとき、実世界の一部が機能する方法のモデルを構築しているのであり、この文脈では、モデルは実際に**適切**かつ**正確**な実世界の構図を表しているときに合理的である。

適切であるためには、モデルは実際に取り扱う世界（システム）の一部でなければならない。つまり、モデルは適切なスコープをもたなければならない。成果に重大な影響を持たない領域を詳細化しすぎたり、一般的すぎたりしてはならない。つまり、重要な詳細がある領域のうわべだけを扱ってはならない。適切であることを保証するため、計画の主要な利害関係者はモデルに影響力を持たなければならない。

正確であるためには、モデル化する因果関係は現実に成立していなければならない。**妥当性基準**（CLR[2]）は**思考プロセス図**の正確性を検証する方法である。CLR は、自他の思考によくある誤りを見つけるのに用いられる。**カテゴリ**（訳注：CLR の「Category」）と呼ばれるのは、明確で少数だからである。**正当**（訳注：CLR の「Legitimate」）と言われるのは、論理を読み書きする人が表明することが常に許されるからである。**懐疑**[3]（訳注：CLR の「Reservation」）と言われるのは、完全には確信がない図の一部を際立たせるからである。懐疑は**常に正当**であるので、誰も感情を害されたと感じることなく、主張し、検討し、理解し、受け入れることができる。CLR は、感情的にならずに理性的でいるための助けとなる。

思考プロセス図を使い始めるときには、図の各部分について CLR をひとつずつ注意深く考えるべきである。しかし、経験を積むにしたがって、CLR を素早く習慣的に適用し始めるだろう。

明快さ

思考プロセス図を自分で作成するなら、表現したいアイディアを持っているのだろう。しかし、いずれ他人と計画を共有する必要に迫られ、その前の最終ステップとして、明

[2] 元の用語は Category of Legitimate Reservation で、意味合いは「正当な懐疑のカテゴリ」である。チェックリストと考えてよい。
[3] チェックリストの個々の項目と考えてよい。本稿では、「懐疑」とする必要がない箇所では「チェック」とした。

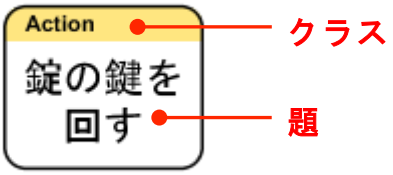
快さをチェックする必要がある。以下を自問しよう。

- 図の各部の意味は明快か
- 図全体として意味は明快か

同様に、初めて見る思考プロセス図を他人が示したときには、以下を自問して明快さの検証を**最初に**行うべきである。

- この図は本当に、この人物が意図していることを伝えているか

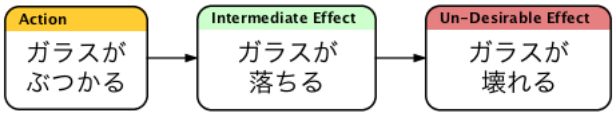
思考プロセス図においては、原因と結果はすべて**エンティティ**として表される。エンティティは、現実を正しく表す、あるいは正しいと思われる短い文を含む四角形である。Flying Logic のエンティティには、上部にエンティティ**クラス**を示す色付きの帯がある。エンティティクラスは、図においてエンティティが果たす役割の種類である。



エンティティの**題**が明快さのチェックをクリアするには：

- 完全で明快であり、文法的に正しい
- 現在形で記述
- 複雑な文でなく、1 つのアイディアを含む点で**単純**

「ぶつかって、ガラスが落ちて壊れた」は、これら 3 つの原理すべてに反する例である。この考えはおそらく、3 つの異なるエンティティとして表すべきである。隣り合うエンティティは因果関係で結ばれる：

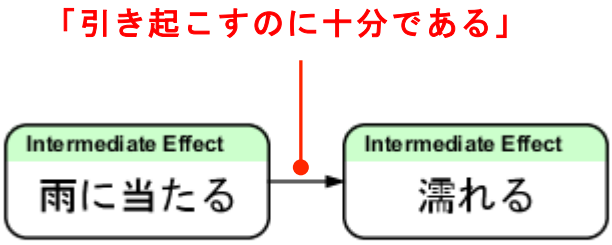


エンティティ間の因果関係も明快でなければならない。エンティティからエンティティへの各ステップは、利害関係者の誰が見ても自然で明らかな流れとなっている必要が

ある。1 つのエンティティから別のエンティティへと**辺**（**矢印**とも言う）を経由して読むことは、2 つのパターン（**思考プロセス**）の 1 つに従う。思考プロセスのどちらを用いるかは図の種類によるが、1 つの図の中では辺の意味は変わらない。

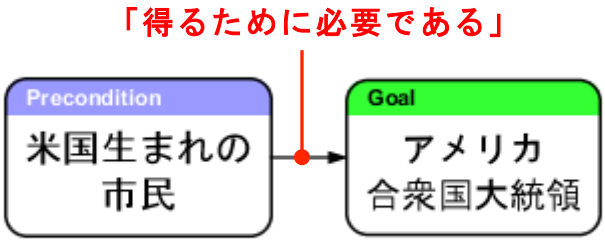
- **十分原因の思考**：「A ならば B である」または「A は、B を引き起こすのに十分である」

このパターンは、A が存在すれば、それだけで、B の存在を引き起こすのに十分であるという考えを表している。十分原因の思考は、**現在ツリー**（CRT）、**未来ツリー**（FRT）、**移行ツリー**（TRT）で用いられる。



- **必要条件の思考**：「A でないならば、B でない」または「A は、B という結果を得るために必要である」

このパターンは、B が存在するためには A が必要であることを表すが、それだけでは十分でないかもしれない。必要条件の思考は、**雲**と**前提ツリー**（PRT）で用いられる。

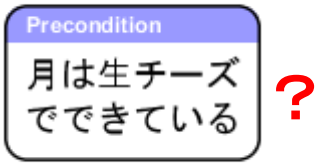


意味は異なるが、どちらの図でも辺（矢印）はまったく同じである。辺をどう読むかは、図を作成するのに用いた思考プロセスによる。

エンティティの存在

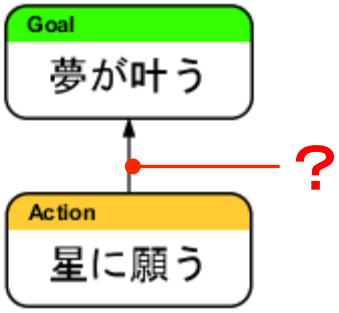
このチェックは、図のエンティティが現在正しいかどうかを問う。たとえば現在ツリー（CRT）においては、図のすべてのエンティティは**現在正しい**何かを記述する必要が

ある。しかし未来ツリー（FRT）または移行ツリー（TRT）においては、現在正しいエンティティと、特定の条件のもとで正しくなると期待されるエンティティとが混在することがある。この懐疑は、実状について正しくない主張をする前に「事実を確認せよ」という警告である。



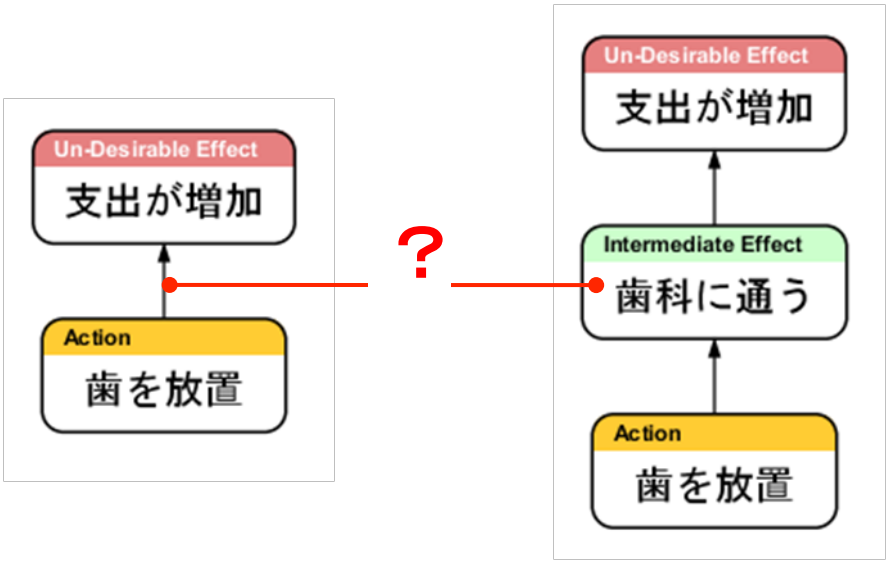
因果の存在/因果の逆転

このチェックは「A は本当に B を引き起こすのか」と問う。2 つのアイディアを、**相互に関連する**という理由で結びつけることがよくある。つまり、両者が近くで見られることがよくあるということである。しかし、あることが別のことを引き起こすことを実際に言うには、遥かに強い証拠が必要である。



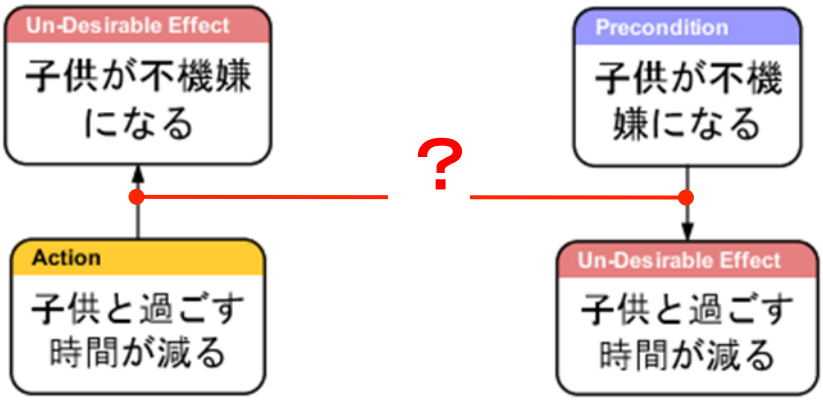
間接的な結果

エンティティが原因の**間接的な結果**ではあるが、重要かつ必要な段階が抜けていることがある。



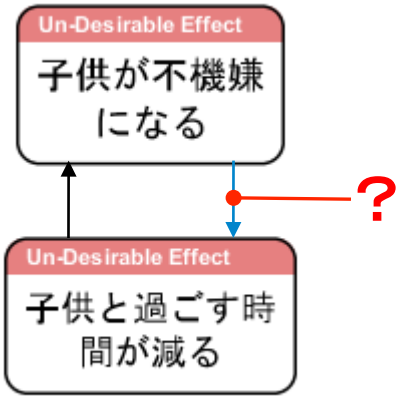
因果の逆転

「因果の存在」の特別な場合が「**因果の逆転**」である。この場合、辺の向きが正しいかを問う。



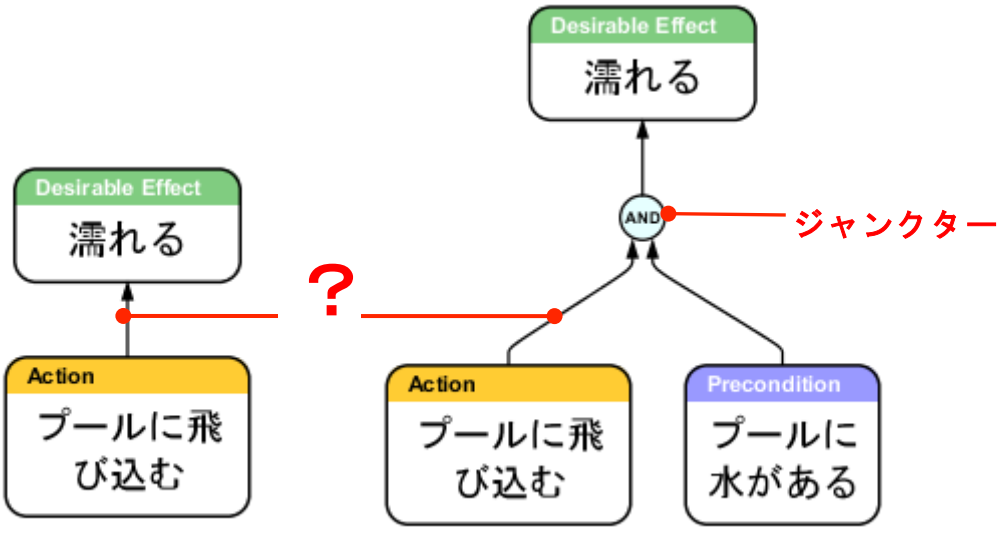
後ろ向きの辺

エンティティが原因であるか結果であるか明らかでない場合、自己強化ループを探すべき場所かもしれない。Flying Logic は**後ろ向きの辺**を用いて自己強化ループをモデル化することができる。エンティティが間接的にそれ自身の原因となる新しい辺を追加しようとしたときは、後ろ向きの辺が自動的に追加される。後ろ向きの辺は、通常の辺よりも太く描かれる。

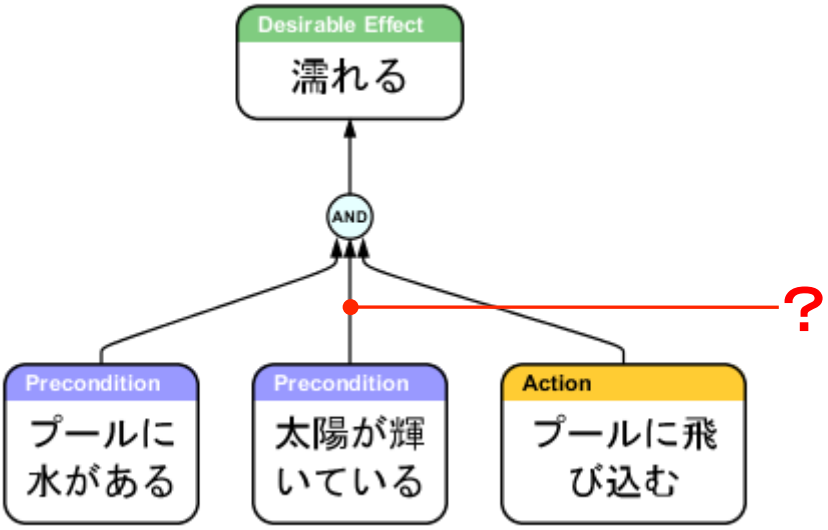


不十分な原因

この懐疑は「B を引き起こすのに A だけで十分か。他に何か必要なのではないか」と問う。特定の結果を生むには、統制できない要因（前提条件）と、影響または統制する要因（行動）の組み合わせが必要であることが普通である。十分条件の思考に基づく図では、AND 演算子を含む**ジャンクター**（接続器）を用いてモデル化する。ジャンクターは、エンティティを既存の辺にドラッグすることで簡単に作成できる。

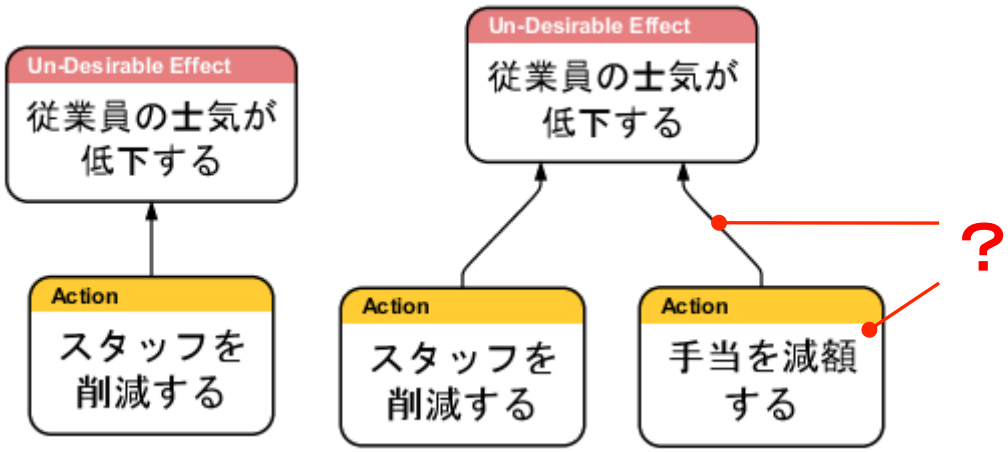


不十分な原因を探すときには、原因が**十分すぎる**こと、つまり結果を生じるには必要でない原因を含むことも念頭に置かなければならない。したがって、「本当は違うのに必要だと列挙したものがないか」とも問わなければならない。



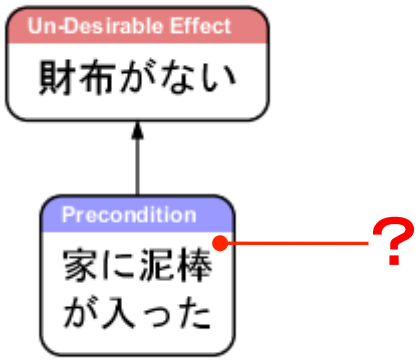
他の原因

ある結果に対して十分な原因を特定したら、先を急ぎたいという衝動に駆られることがよくある。しかしそうすることで、独立にその結果を生じたり、相互に強化し合ったりする他の原因を見逃すかもしれない。このチェックは、「A の原因すべてを特定したか。他に A を発生する原因はないか」と問う。

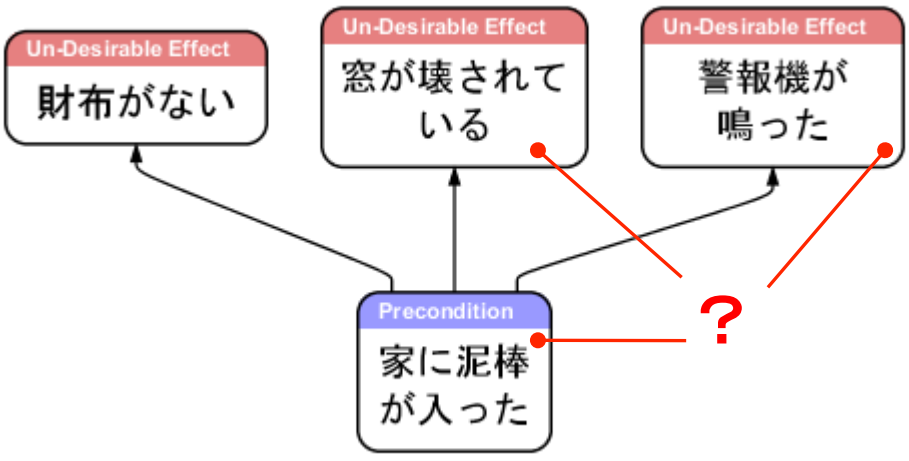


予測される結果

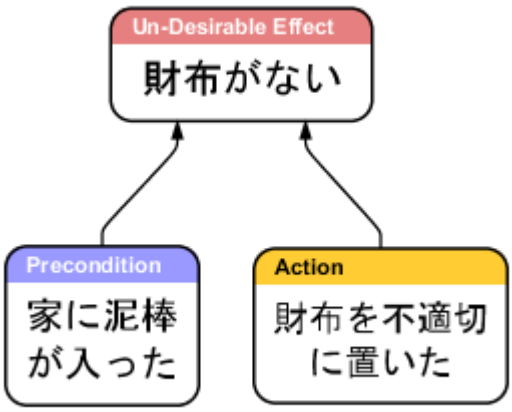
特定した原因が結果を引き起こす**本当の**原因であると信じたいとき、どうすればその確信を深めることができるのだろうか。たとえば、散歩から帰ってきて、財布がないことに気づいたとしよう。初めに心をよぎる可能性のあることのひとつは、家に泥棒が入ったということだ。しかし、泥棒に**入られた**のだろうか。



1 つの原因は 2 つ以上の結果を引き起こすのが普通であり、このチェックは「A が真なら、B に加えて他のどんな結果が予想されるだろうか」と問う。



他の予想された結果も見られるなら、初めに特定した因果関係の確信が深まる。しかし予想された結果が見られ**ない**なら、他の原因を検討した方がよいかもしれない。



トートロジー

自分たちが信じていることをよく吟味してみず、したがって原因を尋ねられたときに違う言葉で言い直すことがよくある。思考プロセス図ではトートロジー（**循環論法**あるいは**論点先取**ともいう）を目にすることはないかもしれないが、日常会話では出会うだろう。いくつか例を示す：

- 「この講義で僕に成績 C をつけることはできない。僕は成績 A の学生だから！」
- 「宿題はつまらない。だって退屈だから」
- 「グリーン市長は一番成功した市長だ。彼は歴史上最良の市長だから」
- 「被告は反省の念を見せない。この事実は、彼を有罪とする決定を強化するのが当然である」

現在ツリー（CRT）

自明でないシステム（たとえば、営利企業、非営利組織、部門、個人的関係）の改善が必要なときには、システムの動作に関する経験が豊富な人にとってさえ、**何を変える**べきか明らかでないことがよくある。システムには複雑に関係する因果関係が多いからである。何を変えるかを決定できるだけシステムを理解することは、より問題であることがよくある。なぜなら、経験豊富な人たちは、システムの自分たちが関わる部分について狭い視野しかもたないことが多いからだ。

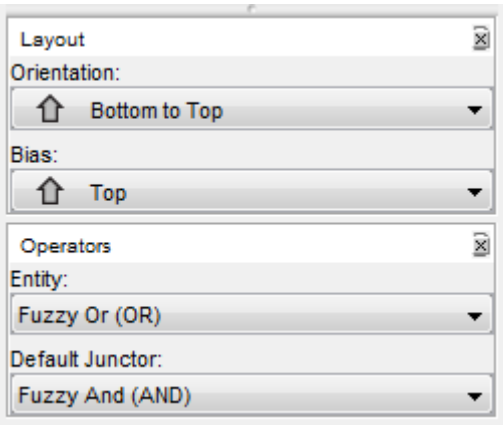
制約の理論（TOC）は、すべてのシステムが**ゴール**、すなわち存在目的をもつという考えに基づいている。システムがゴールを達成する能率は**スループット**と呼ばれる。TOC はまた、すべてのシステムは中核ドライバーをもつとしている。それは、物理的制約、方針制約、市場制約、あるいはこれらの組み合わせの場合があり、システム全体に大きな影響をもち、最終的には（間接的だとしても）システムのスループットを決定する。皮肉にも、システムが**複雑化するほど**、システムがもつ因果関係の相互依存性が増すために、中核ドライバーは**少数になる**。

現在ツリー[4]（CRT）は、システムの中核ドライバー（**制約**とも言われる）を発見するツールである。制約は、システムに生じている**最も深刻**な症状に**共通**する原因である。したがって、スループットを劇的に改善するには、最大限注意を払って制約を管理する必要がある。制約に焦点を当てることで、「投資へのリターン」を最大にできるだろう。

Flying Logic の設定

CRT は十分原因の思考に基づいており、これは初めて作成したときの Flying Logic 文書の設定である。したがって、CRT を作成し始めるのに、Operators（演算子）ポップアップメニューで特別なことをする必要はない。ほとんどの CRT では根本原因を下部、症状を上部に描くので、Orientation（方向）ポップアップメニューを利用して、文書の向きを **Bottom to Top** に変更したいと思うかもしれない。

[4] 訳注：もとの用語は Current Reality Tree である。ここで言う reality とは現実に起きていることを因果関係でモデル化した結果であり、Current Reality Tree は「現在実状ツリー」とも呼ぶべきものであるが、本書では分かりやすさを優先して「現在ツリー」とした。



CRT は、Effects-Based Planning 組み込みドメインに含まれるエンティティクラスを使って作成し、基本的に次のクラスを用いる：Un-Desirable Effect（好ましくない結果）、 Precondition（前提条件）、Intermediate Effect（中間結果）。CRT は問題の特定に最もよく用いられるが、中核の強みを特定するのにも用いられる。この場合には、Desirable Effect（好ましい結果）クラスが用いられることもある。

ステップ 1:範囲を理解する

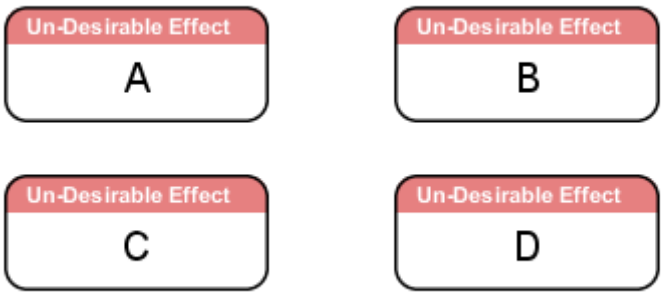
システムがいかに機能し、問題がどこにあるかを文書化する前に、システムについて語るときには自分の意図を明確に理解しなければならない。明確な、書き下された理解に到達するために、利害関係者と必要な時間を使おう。

- システムのゴールは何か
- ゴールが達成されていることを知るための必要条件は何か
- ゴールの必要条件が十分に満たされていることを知るために用いる尺度は何か
- システムの境界はどこにあるか
- システムが一部となっている、より大きなシステムは何か
- システムが相互に作用するシステムは何か
- システムの入力および出力は何か

ステップ 2: 症状を列挙する

おそらく、改善によってシステムに便益が得られるものと考え、種々の問題や障害の症状から潜在的な便益の証拠が分かっているから、分析を行っているのだろう。こうした症状には、低い利益、低い顧客満足、家族間の多くの言い争いなどがある。TOC では、こうした症状は**好ましくない結果**あるいは簡単に **UDE**[5]と呼ばれる。

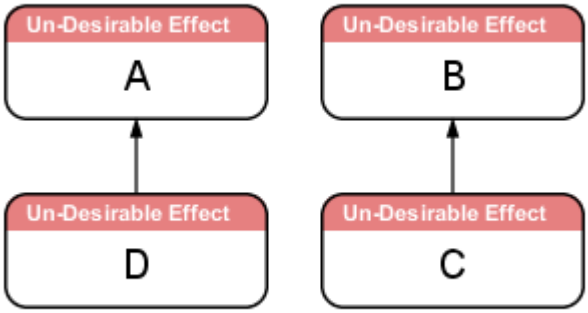
システムの障害のほとんどを引き起こすのは、通常 5 ないし 10 の UDE である。これらを最初に加えなければならない。UDE には利害関係者にとって明瞭かつ簡潔な現在時制の題名をつけ、この段階で選択した UDE は存在について議論がないことを確認すること。つまり、この一覧を見る各利害関係者が「そうだ。これらが、いま抱えている最も深刻な問題だ」と問題なく同意できなければならない。



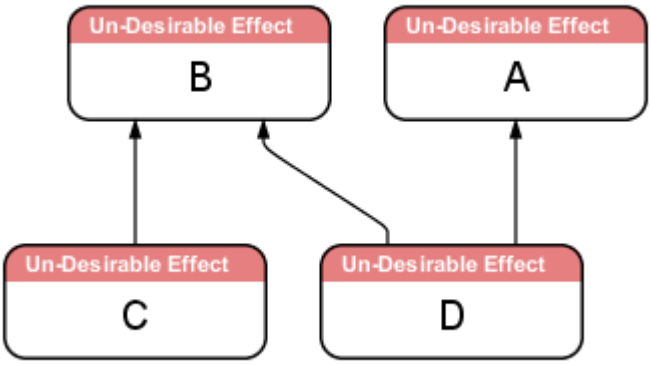
ステップ 3: 症状をつなぐ

好ましくない結果は、他の問題を引き起こすことがよくある。UDE のリストをよく見ると、いくつかは他の結果の直接または間接原因となっていることに気づくだろう。そうであれば、それらのエンティティを原因から結果へと**辺**（矢印）で結ぶ。原因が結果を直接招いているかどうかについて、この段階で悩みすぎないこと。ツリーが大きくなるに従って、全体の構図を完成させる他のエンティティを追加することになる。

[5] 訳注：「ウーディ」と読む。

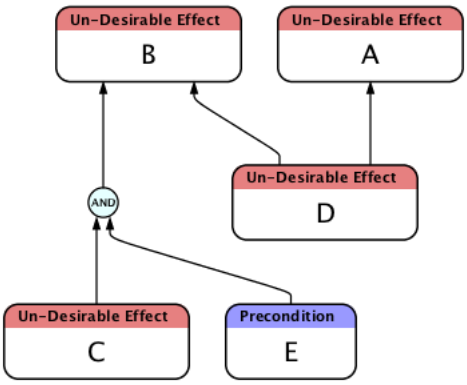


下図の D がそうであるように、1 つの原因が複数の結果を招いていることに気づくことがあるだろう。



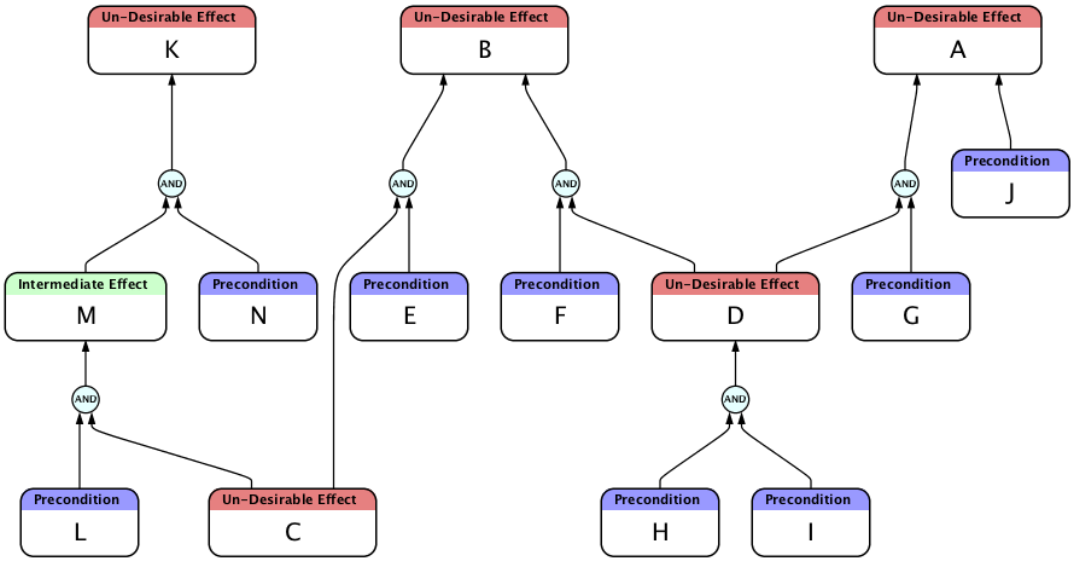
またあるときは、上図の B のように、1 つの結果に複数の独立した原因があることに気づくだろう。Flying Logic 文書が十分原因の思考のために設定されているときには、1 つのエンティティに向かう複数の矢印は、十分かつ独立な原因を意味する。これは、**OR** 関係とも呼ばれる。

1 つの原因が必要だが、それだけでは結果を引き起こさないこともよくある。これを **AND** ジャンクターで表し、原因となるエンティティを既存の辺にドラッグすることで作成する。



ステップ 4:妥当性基準を適用する

図が作成されたとき、おそらくはシステムの非常に粗い構図にすぎないだろう。妥当性基準を適用して、状況の正しい図式を形成するためにエンティティと因果関係を追加する。とくに、特定した結果に対する原因を追加することを検討し、**不十分な原因**を特定して**必要条件**を追加する。また、明快さのために図を吟味して、Flying Logic のコンフィデンススピナー[6]を使って精査する。たとえば、当初 UDE として追加したエンティティが中立的であるとわかったときには、既存のエンティティのクラスを変更することすらできる。



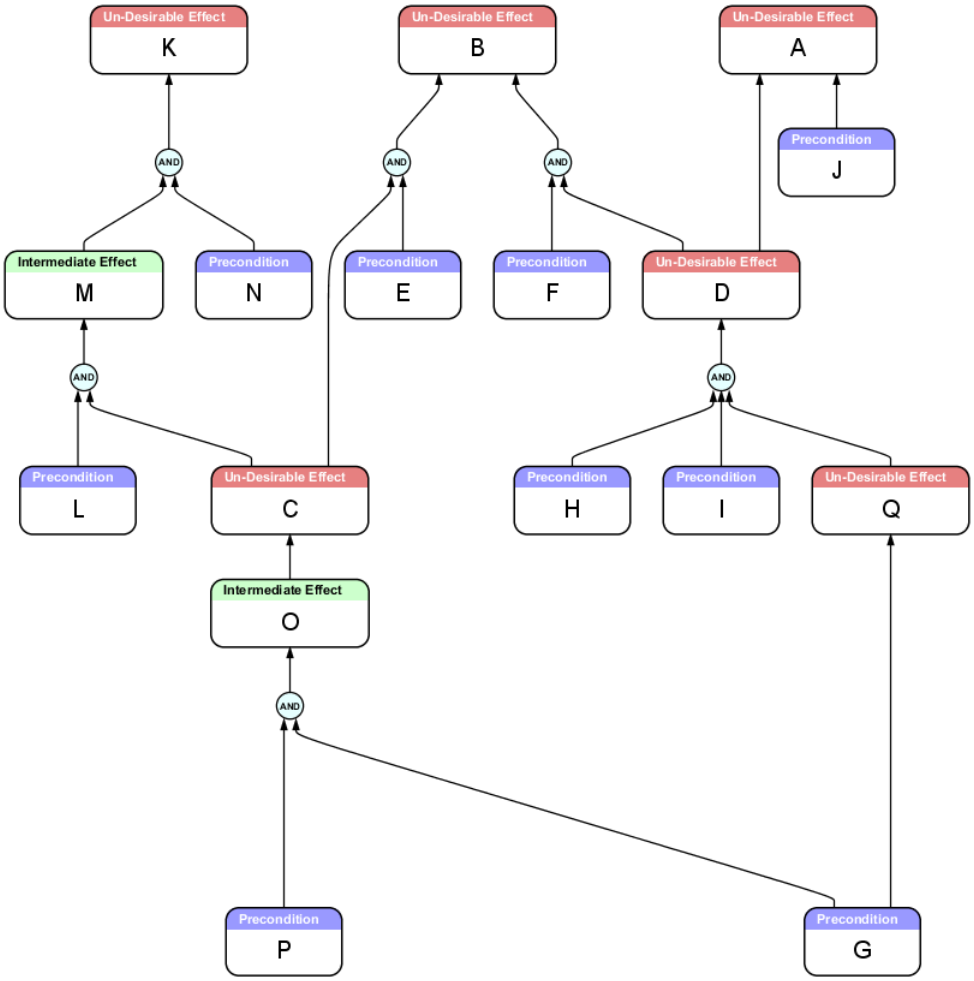
以下のガイドラインは、追加するエンティティのクラスを選択するのに役立つ。

[6] 訳注：確信の度合いを表示および操作できる、円グラフ状のパーツ。

- エンティティが一見して望ましくないなら、言い換えれば、それがなければ確実にシステムがよくなるようであれば、**Un-Desirable Effect** クラスを用いる。UDE は先祖、子孫、またはその両方をもつことがあるが、完成した図では 1 つ以上の因果関係の線をもたなければならない。
- エンティティがネガティブでもポジティブでもなく、システムが置かれている大局的な状況によって存在しているだけであり、それに対して特別重大な影響力ももっていないのであれば、**Precondition** クラスを用いる。Precondition は決して先祖をもたず、常に 1 つ以上の子孫をもたなければならない。
- エンティティがネガティブでもポジティブでもなく、統制下にある何かのために存在するのであれば、**Action** クラスを用いる。Action は常に原因であり、決して結果ではない。したがってそれらは子孫をもつが、先祖を持たない。
- エンティティがネガティブでもポジティブでもなく、図中にある他の原因の結果として存在するのであれば、**Intermediate Effect** クラスを用いる。完成した CRT においては、Intermediate Effect は**常に**先祖と子孫の両方を持たなければならない。

ステップ 5: 背後にある原因の追加を続ける

この段階では、接続されていないエンティティや、緩やかに接続されたエンティティの塊があるかもしれない。このステップでは、とくに 2 つ以上の塊を結びつける原因を求め、図中の結果に対するより深い原因を探して追加する。図の根本に置かれている原因が「なぜ起きているのか」と自問し、答えを追加のエンティティとそれらを結ぶ辺の形にすること。背後にある原因の追加と、前ステップからの妥当性基準の適用を交互に行うこと。

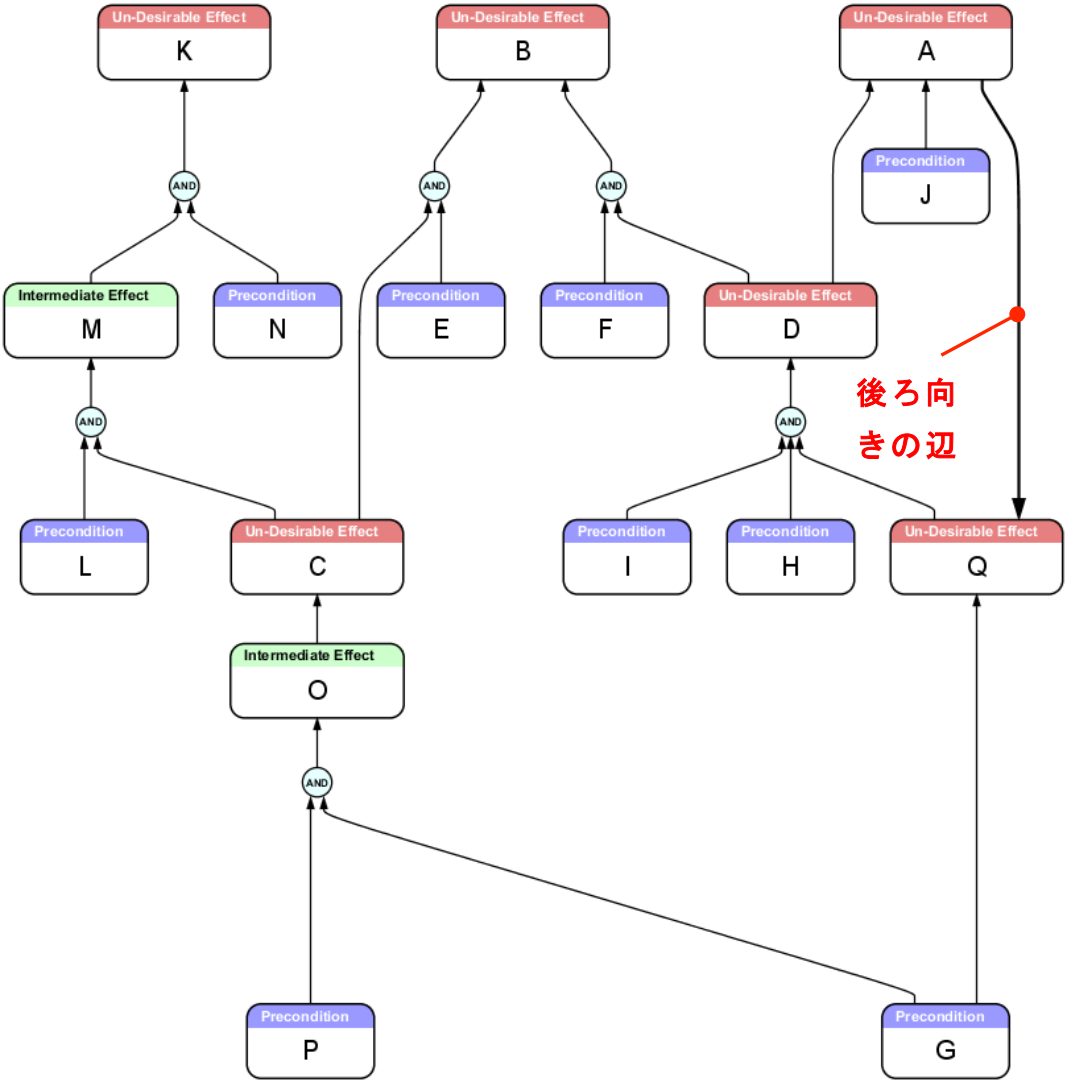


ステップ 6：ネガティブ強化ループを考える

よくあることではないが、システムの上位にある UDE の存在が、下位の UDE に実際に悪影響を及ぼすことがある。この状況は非常に深刻なので、図中では**因果関係ループ**（悪循環とも言う）として記述することが重要である。

通常、Flying Logic 文書のすべての辺は、文書の始点（根本原因）から終点（最終結果）へと「流れる」。ループを形成する(すなわち、結果が間接的にそれ自身の原因となる）辺を追加しようとするときには、Flying Logic は因果関係ループを表す特別な**後ろ向きの辺**を自動的に作成する。後ろ向きの辺は通常の辺よりも太く[7]描かれる。

[7] 訳注：原文では「色が青になる」とも記載されているが、現時点では黒である。



後ろ向きの辺は、通常の辺と 2 つの点で異なる。辺の重みをもたず、文書中のコンフィデンス値の流れに影響しない。しかし、他の辺と同様に、注釈をつけることはできる。

ステップ 7: 根本原因を特定する

CRT が完全になってきたとき、原因のグループが「根本」にあることに気づくだろう。すなわち、子孫はあるが先祖がない。このような原因のいくつかは Precondition であり、他は UDE だろう。そして、それらは図の下部にあるとは限らない。上図では、9 つの根本原因がある。

このステップの目標は、**統制または影響が及ぶ**最も深い原因にたどり着くまでツリーを深掘りしたと確認することである。前提条件は定義により統制外であるが、CRT の

最下部にある前提条件が本当に他の原因の中間結果ではないのかと疑うべきである。また、ツリーの根にある UDE に、統制下にある他の原因がないかも疑ってみるべきである。この考えは、最も深いレベルで問題を「根絶」することである。

ステップ 8: ツリーを刈り込む

ツリーの一部が、関心事の UDE にほとんど、あるいは全くつながっていないことが分かることもある。ツリーを管理可能にするため、これらの塊を以下のいずれかでビューから除去すべきである。

- 削除する
- カットアンドペーストして他の文書に移す
- グループ化して折り畳む

ステップ 9: 中核ドライバーを特定する

原因と結果の規則を遵守して CRT を作成し終えたら、根本原因を除去することで他の問題が除去されていく連鎖反応が起きることも分かるだろう。こうならないと分かったら、図中の各ステップに立ち戻って、UDE の**必要**および**十分**な原因を特定して追加したこと（ステップ 4）、およびツリーを統制または影響範囲の根本原因に至るまで深掘りしたことを確認すること（ステップ 5）。

CRT において最も重要な UDE に対して最も影響する、1 つの原因を特定するときが来た。この原因は中核ドライバー（**制約**または**ボトルネック**とも呼ぶ）である。システムを低下させる境界を打ち破るために、これを管理または除去しなければならない。

CRT には、最終的にはすべての原因に注意を払わなければならない、根本原因が複数あるかもしれない。根本原因に対して要因を判断することにより、中核ドライバーを見つけることができる：

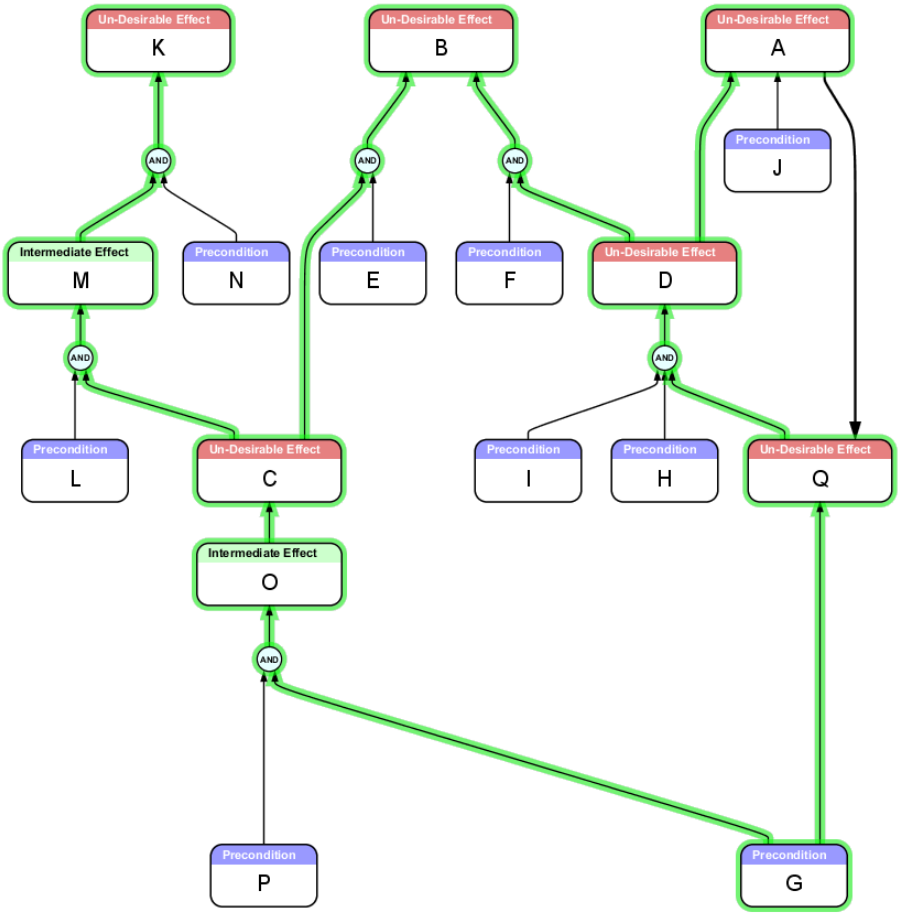
- **どれだけの** UDE を間接的に引き起こすか
- 間接的に引き起こす UDE は**どれだけ深刻か**
- それらを**どれだけ統制または影響できるか**

Edit ➧ Cut/Copy Includes Successors スイッチをオンにして、根本原因の 1 つを選択し[8]、**Edit ➧ Copy** を選択する。これにより、選択した原因と、直接および間接の

[8] 訳注：原文では Flying Logic Pro の機能として述べられている。現在は他のエディションがないので、

先祖がハイライトする。（**エスケープ**キーを押下すると、ハイライトをキャンセルできる。）UDE の重要性を考慮する必要は変わらないが、このテクニックによって根本原因がどれだけの影響をもつかを素早く確認できる。

下図では、**Includes Successors** スイッチをオンにして、エンティティ G を選択およびコピーしている。これによって、G がある意味で図中すべての UDE に影響していることが明らかである。図中すべての原因が少なくとも影響下にあるので、G が中核ドライバーであると結論する。つまり、これが焦点を当てるべき制約である。



当該部分を訳さなかった。

雲 — 対立の解消

論争。紛争。政治。敵。妥協。損失。

誰しも対立に出合ったことがあり、ほとんどが、可能であれば対立を避けようとしている。対立は不健康で不愉快であると見られている。対立の無視が状況を悪くする可能性に気づいていてさえ、対立を無視しようとする人が多いほどである。さもなければ、どちらの側も対立をゼロサムゲームとして扱っている。「自分がやるか、そうでなければ彼がやる」というように。または、恐らくさらに悪いことに、双方が妥協する。対立は「赤ん坊を裂く」ので、誰も幸福にならない。

対立を解決するよい方法があることが分かっている。関係者全員を完全に満足させる解決策を創造する方法である。ある驚くべき視点からは、**立場**という表層的なレベル—何を望むと言うか—を除けば**対立は実在しない**と考えることさえできる。

2 つの欲求が相容れないとき、対立すると言う。前に進むには、**欲求**（**立場**とも言う）が背後にある**要求**（**要件**とも言う）に動機づけられていると理解する[9]ことである。たとえば、玩具をめぐる子供の喧嘩から 2 つの欲求が生じることがある。この場合、二人とも 1 つの限られた資源を独占したいという**欲求**をもっている。しかし、二人は背後にある**要求**に動機づけられている。二人の欲求は同じではないかもしれないが。一人は玩具の所有権を主張したいかもしれないし、もう一人は玩具を遊びに取り入れたいだけかもしれない。さらに、子供たちには共通の目的がある。仲良くして、楽しく過ごすことだ。この**共通目的**を達成するには、どちらの要求も満たす必要がある。子供たちの欲求から生じた明らかな対立の境界を越えて要求と共通目的を認識したので、創造的な解決策への扉が開いたことに注意して欲しい。子どもたちが思っているよりも、二人ともハッピーなれるかもしれない。

ある種の異常ではなく**正当な要求**と**共通目的**から対立しているという因果関係がわかれば、最良のアプローチは逃避ではなく、速やかなコミュニケーションと互いの利益になるオプションの創造であることが明白になる。

現在ツリー（CRT）の作成後、中核ドライバーを 2 つの排他的な立場からなる**中核**

[9] 訳注：欲求は want、要求は need を訳出した。マーケティングの考え方を思い起こすと、理解しやすいだろう。

の対立に再構成できることが多い。対立する欲求（CRT を作成した結果として生じることもあるが、独立に生じることの方が多い）に直面したときはいつでも、「**雲**[10]」がまさに使うべきツールである。（対立を「蒸発させる」能力から、そう呼ばれる。**対立解消図**とも言う。）

Flying Logic の設定

雲の基本形は常に同じなので、最も簡単な作成方法は **Examples/Conflict Resolution** フォルダにある組み込みテンプレートファイル **Cloud.logic-t** を開くことである。テンプレートファイルは、誤ってテンプレートファイル自体を変える恐れなしに変更および保存できる、無題の新しい文書として開かれる。

次の段落で、新しく雲の文書を作成する準備を説明する。テンプレートを使用する場合は、次のセクションへ進んでよい。

雲は必要条件の思考に基づいている。Flying Logic 文書はデフォルトでは十分条件の思考向けに設定されているので、Operator（演算子）ポップアップメニューを次のように設定したいだろう。

- Entity Operator: Fuzzy And (AND)
- Default Junctor Operator: Fuzzy Or (OR)

雲は、Common Objective（共通目的）から始めて、左から右に読む。しかしこれは、辺（矢印）が Common Objective に向かう、つまり右から左への向きでなければならないことを意味する。したがって Orientation（向き）ポップアップメニューを **Right to Left**（右から左）に設定したいだろう。

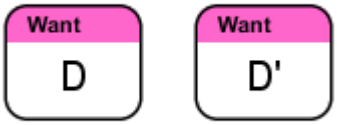
[10] 訳注：元の用語は Evaporating Cloud で、「蒸発しつつある雲」という意味合いである。原文では単に Cloud と呼ぶこともあり、本書では「雲」で統一した。なお、文献によっては、EC という略称を用いることもある。

雲は、組み込みドメイン Conflict Resolution（対立解消）のエンティティクラスを用いて作成し、次のクラスを使う：Want（欲求）、Need（要求）、Common Objective（共通目的）、Conflict（対立）、Solution（解）

前述のテンプレートを使っている場合には、図はすでに描かれている。テキストを記入するだけでよい。しかし、以下では雲を始めから作成することを前提としている。

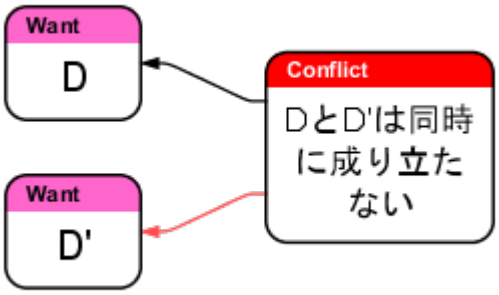
ステップ 1：欲求を特定する

2 つの **Want** エンティティを作成し、対立する立場のそれぞれを簡潔に要約する題名を記入する。これらの Want は、習慣的に D と D'（D プライム）と呼ばれる。

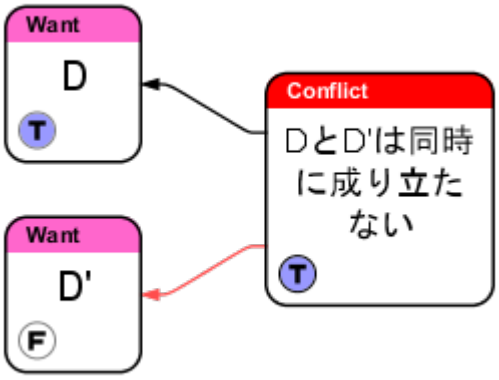


ステップ 2：対立を特定する

ひとつの **Conflict** エンティティを作成し、2 つの Want それぞれの先祖とする。雲の典型である右から左への向きにしているのなら、Conflict エンティティは Want の右になる。Conflict エンティティに、Want が対立する**理由**を要約する題を記入する。



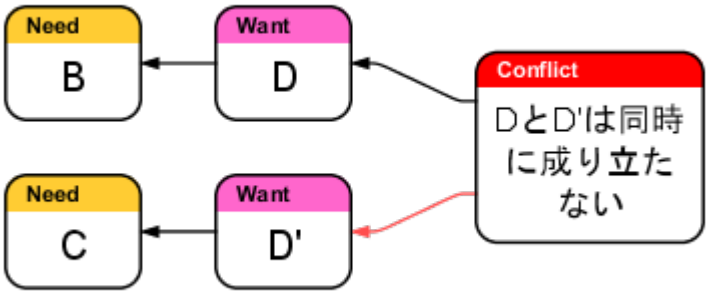
最後に、Conflict エンティティから出ている 2 つの辺の 1 つを右クリック（Mac、Windows）またはコントロール+クリック（Mac）して、現れたポップアップメニューの **Negative**（否定）を選択する。これによって、否定した辺が赤くなる。こうすることにより、モデルが 2 つの欲求の相互排他な性質を正確に反映する。これを確認するには、ツールバーの Show Confidence Spinners（コンフィデンススピナーを表示）スイッチをクリックし、Conflict エンティティのスピナーを最大値（True）から最小値（False）へ変化させる。否定された辺によって 2 つの Want エンティティが同時に真になり得ない様子が分かる。



ステップ 3: 背後にある要求を特定する

2 つの **Need** エンティティを作成し、それぞれを Want エンティティの子孫とする。立場の目的は、背後にある要求を満足することである。各 Need に、対立の一方が立場（Want）の表明によって満たそうとしている、直接の要求を要約した題名を記入する。

Need と Want の差異は単純である。Need の充足は全体の目的の充足に必要な**条件**だと見なされるが、Want は要求を充足するよう選択された特定の**行動**である。

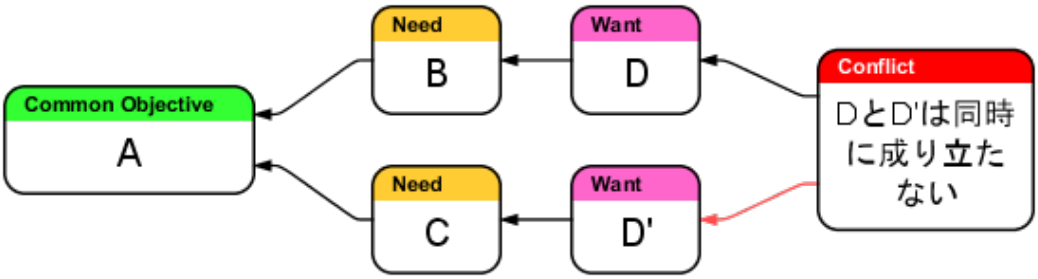


ステップ 4: 共通目的を特定する

1 つの **Common Objective** エンティティを作成し、2 つの要求双方の子孫とする。右から左への向きでは、Common Objective エンティティは図の最も左になる。

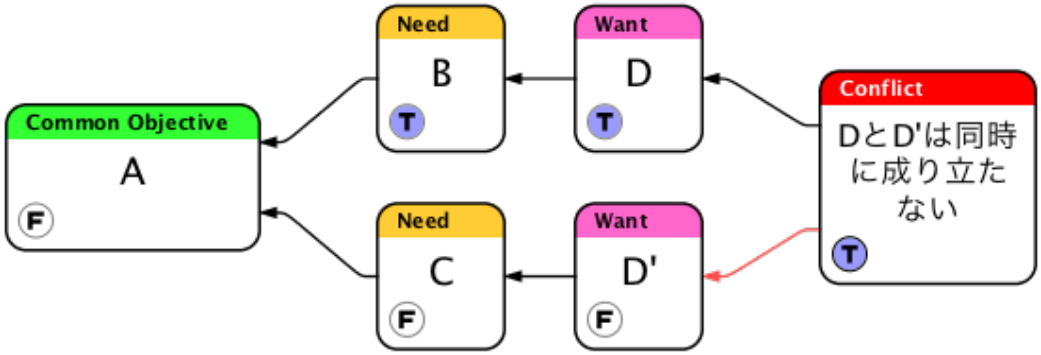
対立の両側が共通の目的をもたないのであれば、両者は単に別々の道を進むことができる（協調する理由がない）ので、実際には対立が存在しないことになる。このように、対立と特定されたすべての状況においては、共通目的が常に存在する。直前のステップで特定された要求は、いずれも共通目的の達成に必要であると見なされる。言い換えれば、対立する両者は、自らの要求が満たされなければ共通目的が満たされないと確信している。

Common Objective はまた、Want や Need より「上位」にあるのが普通である。玩具を巡る子供たちの喧嘩の事例では、彼らの Common Objective は「仲良くして、楽しく過ごす」ことだろう。Want と Need のすべてが対立の原因である特定の玩具について言及していたとしても、この Common Objective がそれに言及していないことに注意する。



ステップ 5: 図を読み上げて明快さを確かめる

図が完成したので、コンフィデンススピナーを表示する。唯一のドライバー（Conflict エンティティ）が存在することに注意する。これは、先祖がない唯一のエンティティである。上述のように文書演算子を設定して Conflict エンティティから出ている辺の 1 つを否定していれば、Conflict エンティティのスピナーの値を変更することにより、どちらか一方の Want を満たす（**True** になる）ことができるが、Common Objective は決して **True** にできないことが分かるだろう。言い換えれば、対立が存在する限り、Common Objective は達成できない。



図を読み上げて、明快さと正確さを改善しよう。雲は、以下のパターンを用いて、辺の流れとは逆に左から右に読む。

- 「**要求を充足**させるためには、**欲求を満足**しなければならない」

これは、必要条件の思考の基本パターンである。このパターンは、Conflict エンティティから出ている 2 つの辺を除く全ての辺に適用される。このステップを終えたら、表現を修正または明確化する。

ステップ 6: 仮定を特定し検証する

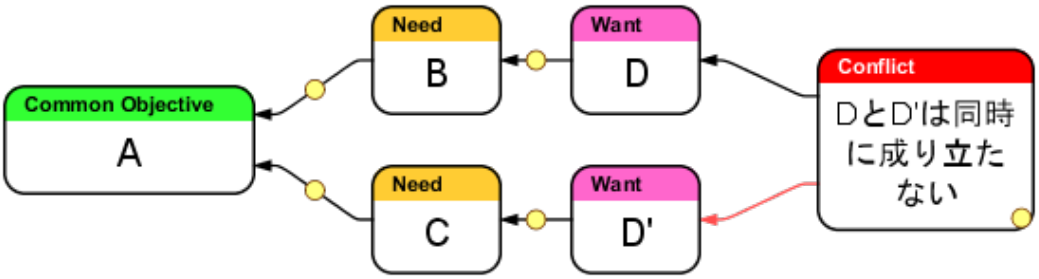
直前のステップからのパターンにおいては、要求と欲求に対して 2 つの空欄がある。このステップでは、3 つめの空欄を追加する。

- 「**要求を充足**させるためには、**仮定により**、**欲求を満足**しなければならない」

仮定は欲求を満足しなければならない**理由**であり、誤った仮定を見つけることが対立を解消する鍵である。仮定は Want→Need の辺および Need→Common の辺の下に「隠

れて」いる。Conflict エンティティ自体にも、仮定が潜んでいる。両方の Want を同時に満たせないと信じている**理由**はなにか。

これらの仮定に出会ったら、Flying Logic の注釈機能を使って、4 つの辺のそれぞれと Conflict エンティティに文章を追加する。仮定を記述するのに必要なだけの文字数を使い、それぞれ「なぜなら」で始める。



各辺に 1 つ仮定がある場合もあるが、いくつかあることも多い。仮定は**妥当**か**妥当でない**かのいずれかである。誤った仮定によって欲求と要求を結びつけることがよくあるが、ここでは各前提を批判的に評価して妥当性を判断しなければならない。誤った仮定を除去し、正しいものだけを残すことができる。

ステップ 7: 解決策を提案する

辺の**いずれか**にある**すべて**の仮定を無効化できるなら、2 つのエンティティ間にある必要条件の関係を除去したことになる。Conflict エンティティのすべての仮定を無効化したなら、対立の認識自体を除去したことになる。どちらの場合であっても、雲は「蒸発した」のであり、実際にもう対立がないことを確認したのである。

雲がまだ残っていれば、5 カ所のうち少なくとも 1 つは正しい前提がある。最後のステップは、雲の 1 つ以上の辺を「壊す」解決策を構築する（注入[11]とも言う）ことである。解決策は「相互の利益となる選択肢」であり、創造的な解決策を見いだす最も建設的な場所は Want と Need を結ぶ辺である。以下のパターンを問う。

- 「どうしたら Want を受け入れずに Need を満たせるか」
- 「どうしたら Need を満たすことなく Common Objective を達成できるか」

[11] 訳注：通常は solution であるが、TOC では injection と呼び、邦訳ではそのままカナ書きしてインジェクションとする場合が多い。しかし、読み手に取って意味が明瞭でない語をカナ書きしても、不明な用語を増やすことに過ぎないので、本書では「注入」としている。なお、Flying Logic で注入するエンティティクラスは Solution である。

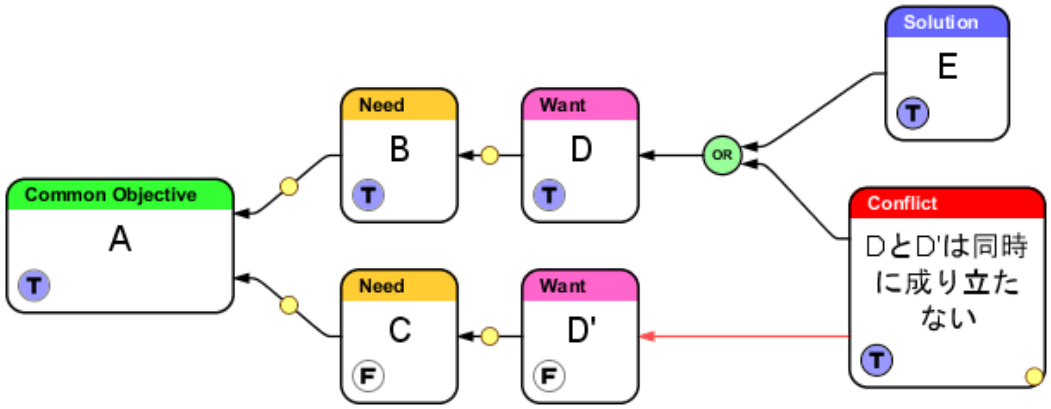
- 「どうしたら**最初の** Want と **2 番目の** Want の両方を受け入れられるか」

分析している状況がかなり簡単でない限り、この段階で得られた解決策を最終版と考えてはならないことを覚えておこう。このツールは、新しい選択肢を得るブレーンストーミング用である。この段階で得た考えを強化するために未来ツリー（FRT）を用いることができる。また、1 つの「万能薬」的な解決策を探したい誘惑を避けること。真に Win/Win の解決策を実現するためには、2 つ以上の注入が必要となることが多い。

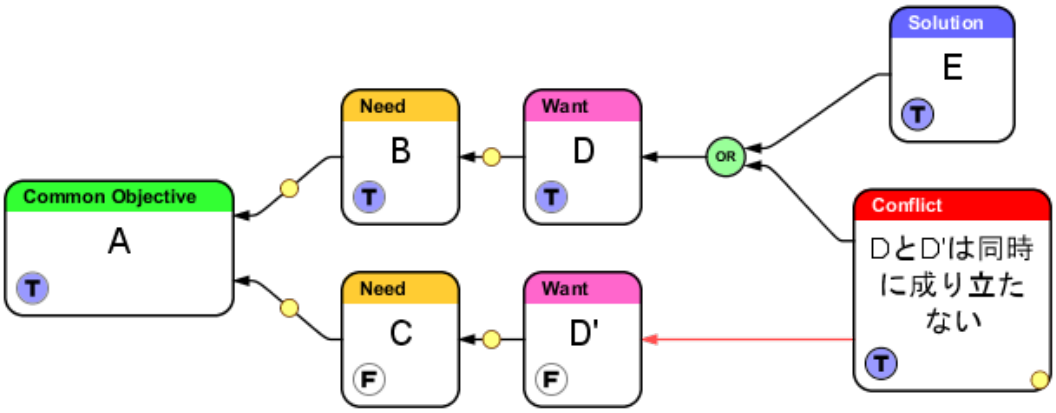
解決策を図に注入するには、初めに解決策を出現させたい辺を選択し、次のいずれかを行う。

- サイドバーにあるクラスの一覧で Solution エンティティをダブルクリックする。
- 辺の上で右クリック（Mac、Windows）またはコントロール+クリック（Mac）し、現れたポップアップメニューから **Solution** を選択する。

OR ジャンクターが辺に挿入され、新しい Solution エンティティが先祖として追加される。この新しいエンティティに、解決策を要約した題名を記入する。



そのジャンクターだけを選択し、ポップアップメニューの同じコマンドを用いることで、さらに解決策を追加できる。コンフィデンススピナーを表示して、追加した各解決策に対する他の制御点があることと、たとえ両方の Want（もとの立場）を満足しなくとも、Common Objective を **True** にできることを確認できる。


Common Objective
A
T
Need
B
T
Need
C
F
Want
D
T
Want
D'
F
OR
Solution
E
T
Conflict
DとD'は同時に成り立たない
T

未来ツリー（FRT）

おそらく読者には改善したいシステムがあり、何を変えるかを特定するために現在ツリー（CRT）を作成し終えたのだろう。いくつかの潜在的な Win/Win 解決策、言い換えれば**何に変えるか**を特定するために、1 つ以上の雲も作成し終えたかもしれない。しかし、

- どうやって、どのアイディアが有効かを確信できるか。
- どうやって 1 つのアイディアを選択するか。
- どうやって何か重要なことを無視あるいは見逃していないことが分かるか。
- 解決策の強みが何で、どうすればそれを最大化できるか。
- 「解決策」が状況を今よりも悪化させる予期せぬ結果を招かないことを、どうやって確信できるか。
- 解決策の潜在的な欠点は受け入れられるものか、事後に解決できるものか、あるいはいかなる代償を払っても避けるべきものか。

これらの問いに答えられるよう支援するのが、未来ツリー（FRT）の役目である。

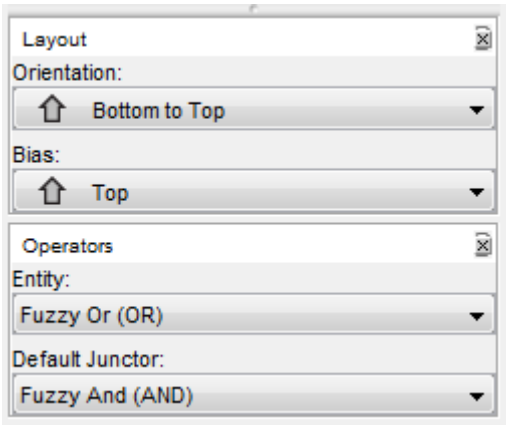
現在ツリー（CRT）と比べると、FRT を簡単に理解できる。

- CRT を構築するには、好ましくない結果（UDE）から始め、解決策（注入とも言う）を考案した箇所である中核ドライバーへと下向きに構築する。
- FRT を構築するには、潜在的な解決策（**注入**）から始め、好ましい結果（DE）の集合へと上向きに構築する。

FRT は、すでに認識している解決策からだけでなく、すでに作成した CRT や雲から構築することもできる。

Flying Logic の設定

FRT は十分原因の思考に基づいており、これは初めて作成したときの Flying Logic 文書の設定である。したがって、FRT を作成し始めるのに、Operators ポップアップメニューで特別なことをする必要はない。ほとんどの FRT では提案された解決策を下部、好ましい結果を上部に描くので、Orientation ポップアップメニューを利用して、文書の向きを **Bottom to Top** に変更したいと思うかもしれない。



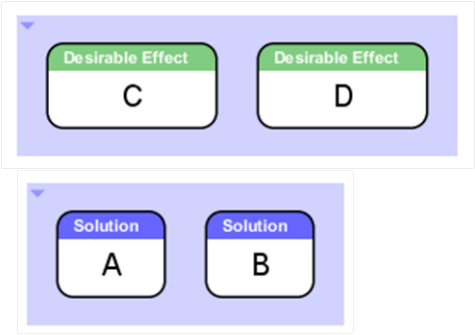
FRT は、組み込みの"Effects-Based Planning"ドメインに含まれるエンティティクラスを使って作成し、基本的に次のクラスを用いる：Desirable Effect（好ましい結果），Precondition, Intermediate Effect, Action （アクション)。雲で得られた解決策から始める場合は、Conflict Resolution ドメインの Solution クラスを FRT で使うこともよくある。

ステップ 1: 提案された解決策と好ましい結果を記述する

ひとつ以上の Solution エンティティを作成し、実現しようと計画している注入の集合を特定する。これらの注入は作成済みの雲から得られることもあり、Copy および Paste コマンドを使って、FRT 文書にこれらを容易に追加することができる。

作業中の成果を要約した、1 つ以上の好ましい結果も作成する。

これら 2 組のエンティティを分離しておくために、一時的なグループを作成したいと思うかもしれない。FRT の目的は、これらの「間」を埋めることである。

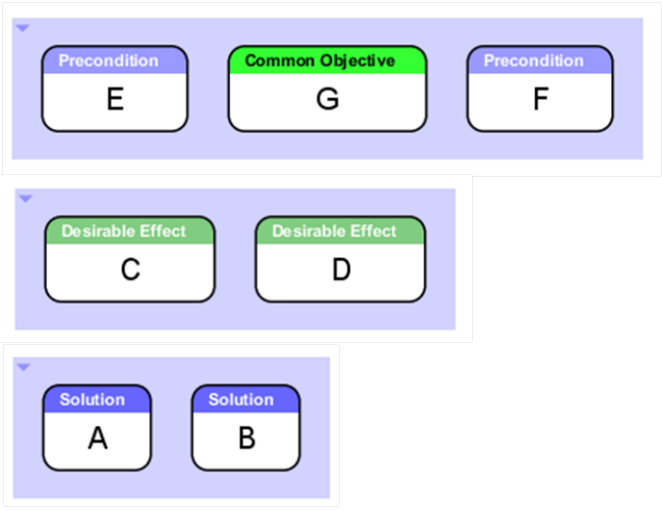


ステップ 2：作成済みの他のエンティティを追加する

CRT を作成済みであれば、FRT で必要になるかもしれない Precondition エンティティ（既存の実状に関する文）を探す。Copy および Paste コマンドを使って、CRT から FRT へそれらを簡単にコピーできる。

既存の雲を使っていれば、Common Objective、および提案された解決策が充足を目指す Need エンティティもコピーする。

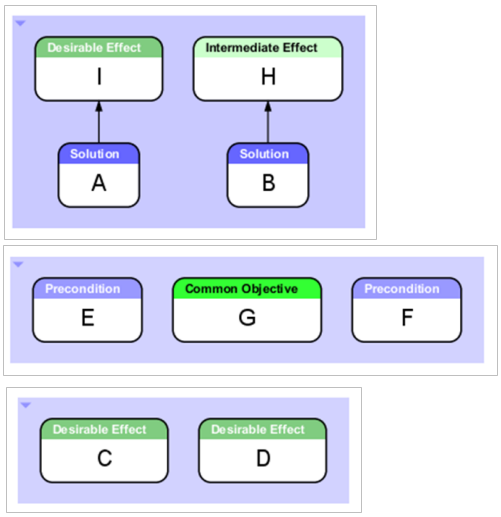
このステップで追加したエンティティは FRT の「間」となるので、これらをグループ化したいと思うかもしれない。



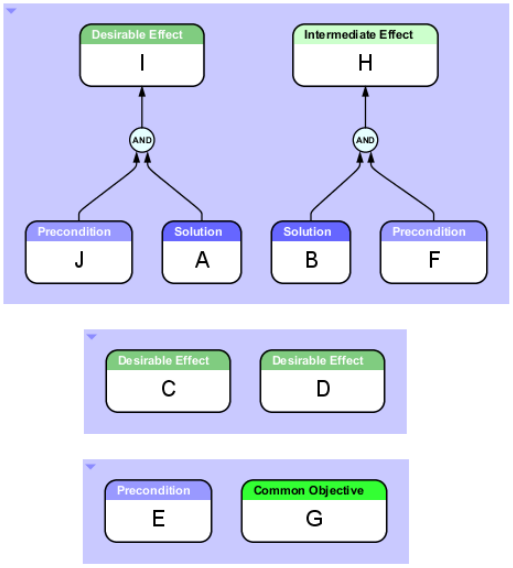
ステップ 3：ギャップを埋める

Solution エンティティから始めて、導入した注入の直接的かつ不可避な結果を表すエンティティを追加する。ネガティブな結果に対しては Un-Desirable Effect エンティティ、ポジティブな結果に対しては Desirable Effect エンティティ、中立的な結果に対し

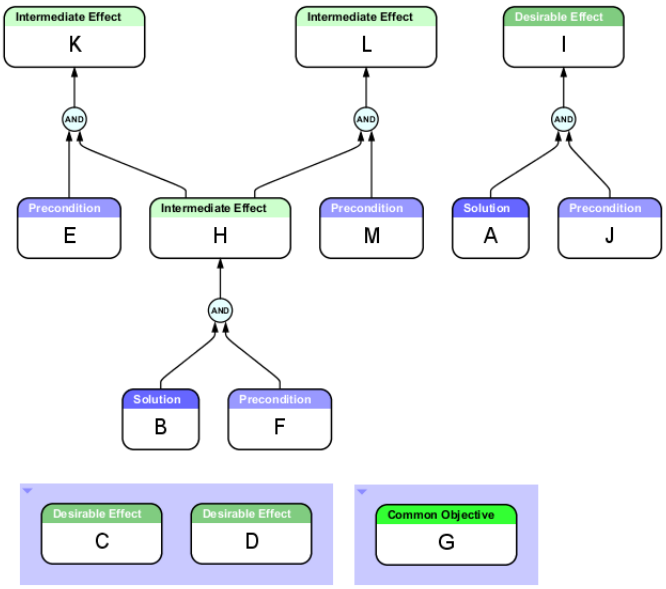
ては Intermediate Effect エンティティを用いる。



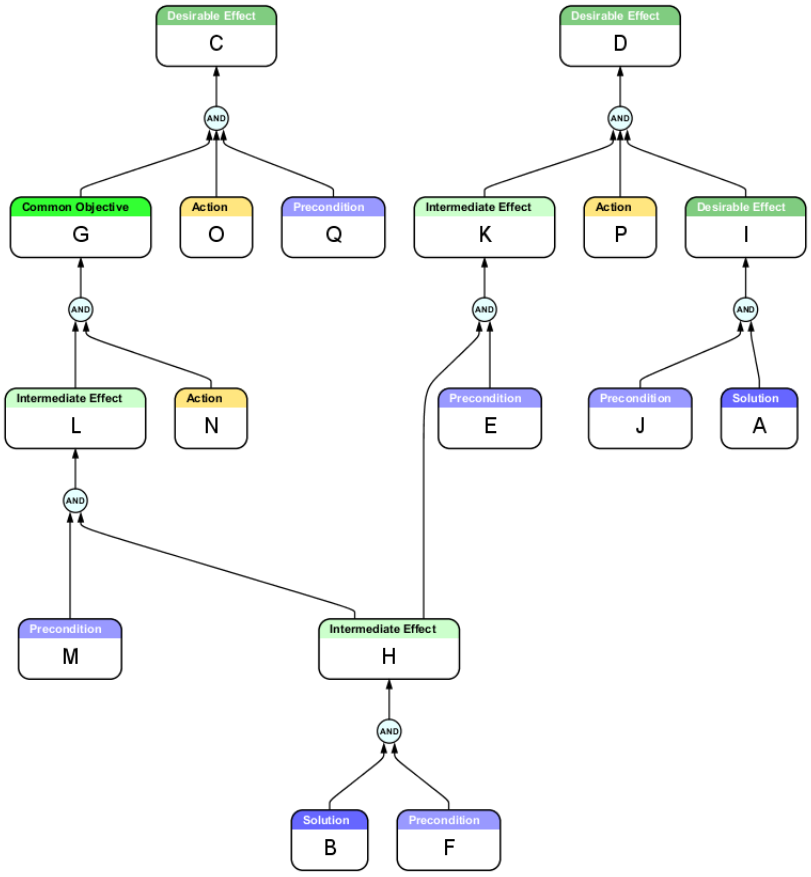
因果関係をチェックするために、妥当性基準を用いる。追加した結果がそれ自体では十分でないなら、Precondition エンティティを追加するか、予測される結果を生じるのに必要な他の必要条件（AND 関係）を表す他のエンティティと結びつける。辺によって文書が構造をなし始めたら、オブジェクトをグループ間で移動したり、エンティティを非グループ化したり、自由にしてよい。



特定した結果から他の結果へと作業を進める。このとき、引き起こされる結果により、Desirable Effect、もしくは雲から追加した Common Objective または Need へと近づいているかどうかを評価する。もし近づかないなら、以前検討しなかった結果の追加と評価を続ける。



進捗が低下または停止したら、Action エンティティの追加を検討する。これらの Action も注入であるが、もとの解決策の注入と区別するため、作業の出発点とした Solution エンティティではなく Action エンティティを用いる。



ステップ 4：ツリーを読み、検証する

すべての Desirable Effect への関係を構築し終えたら、もう一度、図全体を読む。FRT が十分原因の思考で作成されたことを思い起こすと、読むときの基本パターンは次の通りである：

- **原因** A ならば**結果** B である

1 つのエンティティに 2 つ以上の矢印が入るなら複数の十分原因があり、これは OR 関係とも呼ばれる：

- **原因** B または**原因** C ならば**結果** D である

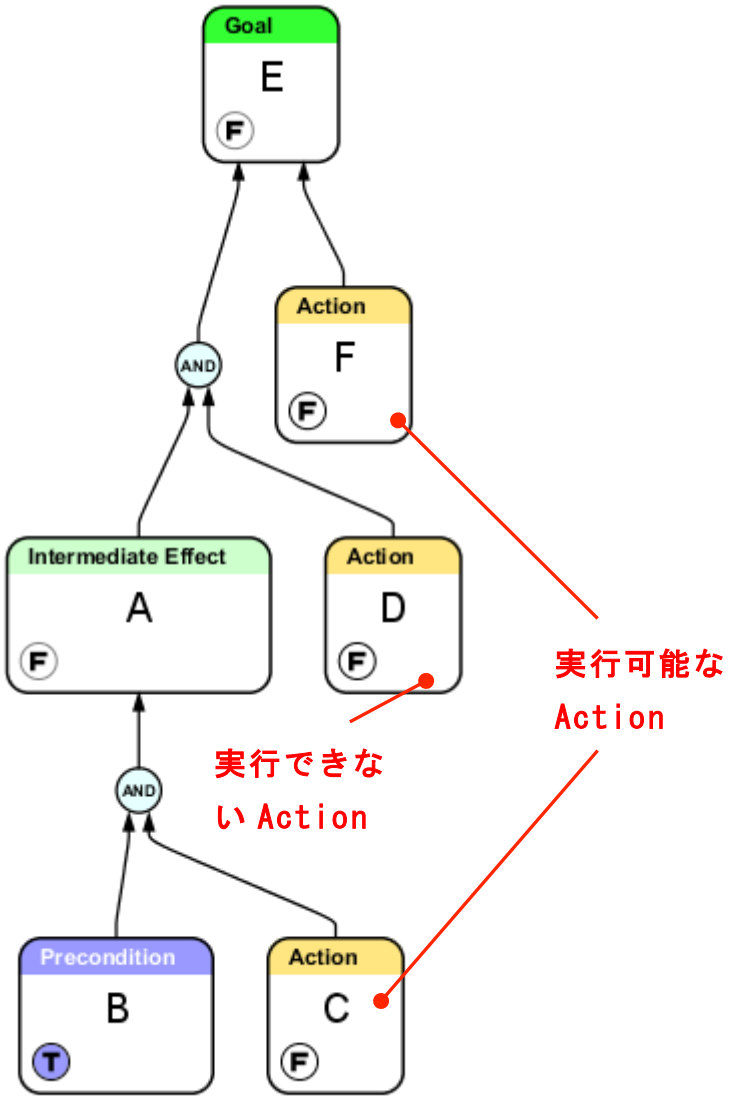
2 つ以上の矢印が 1 つの AND ジャンクターに入るなら、**複数の必要条件**があることになる：

- **原因** E かつ**原因** F ならば**結果** G である

妥当性基準に注意を払うこと。エンティティ中のすべての文と、因果関係が意味する文とが明快で論理的であることを確認すること。

図を読むときには、コンフィデンススピナーを表示して、それらを論理の検証に役立てるとよい。

1. ツールバーの **Confidence** ボタンをクリックするか View→Confidence コマンドを選択して、コンフィデンススピナーを表示する。
2. Entity→Reset Confidence コマンドを用いて、各スピナーを**不確定**に設定する。
3. Precondition は現状の事実と考えられるので、各 Precondition エンティティのコンフィデンス値を True に設定する。
4. Solution 自体が結果の原因として十分であることを認識し、そのコンフィデンス値を True に設定して、それらの結果が True となることを確認する。
5. いくつかの Action が AND ジャンクターで、現在 True である他のエンティティと「結合」されていることを確認する。これらの Action は**実行され得る**が、まだ True になっていない他のエンティティと結合されている他の Action は**実行され得ない**。それらは、他の必要条件が True になるまで待たなければならない。

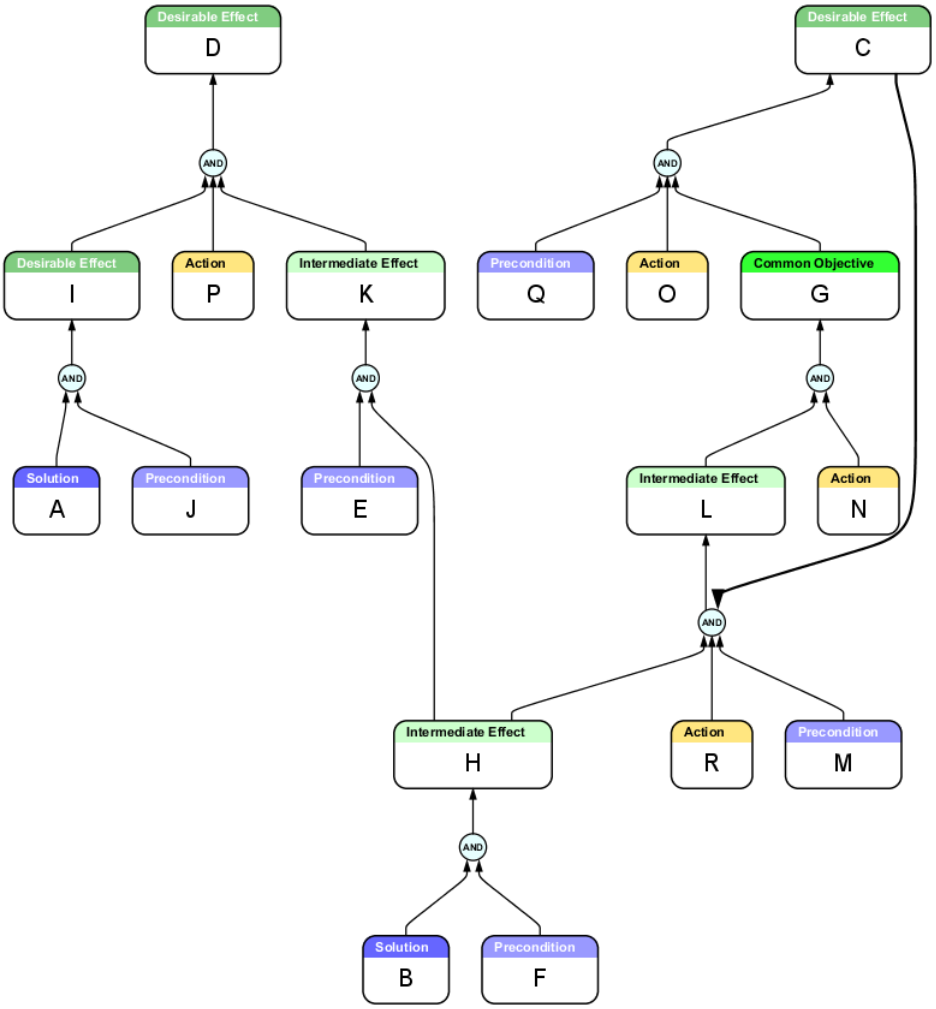


6. 各 Action が実行可能となって True を設定するときには、自分自身に図の「物語」を語りながら、Desirable Effect も True になるまで、図全体に渡ってステップごとに作業を続ける。途中で見つけた論理の誤りは訂正する。

ステップ 5:ポジティブな強化ループを構成する

現在ツリー（CRT）を作成したときに、非常に深刻な、好ましくない結果が他に「フィードバック」し、ネガティブな強化ループを形成する場合があったことを思い出そう。FRT を作成するときには、正反対のことを行う機会に対してループすること、つまり**ポジティブ**な強化ループを構成することを望む。そうできる場合には、自立した解決策を構築できる可能性が高まる。1 つ以上の複数の好ましい結果に影響している、ツリー

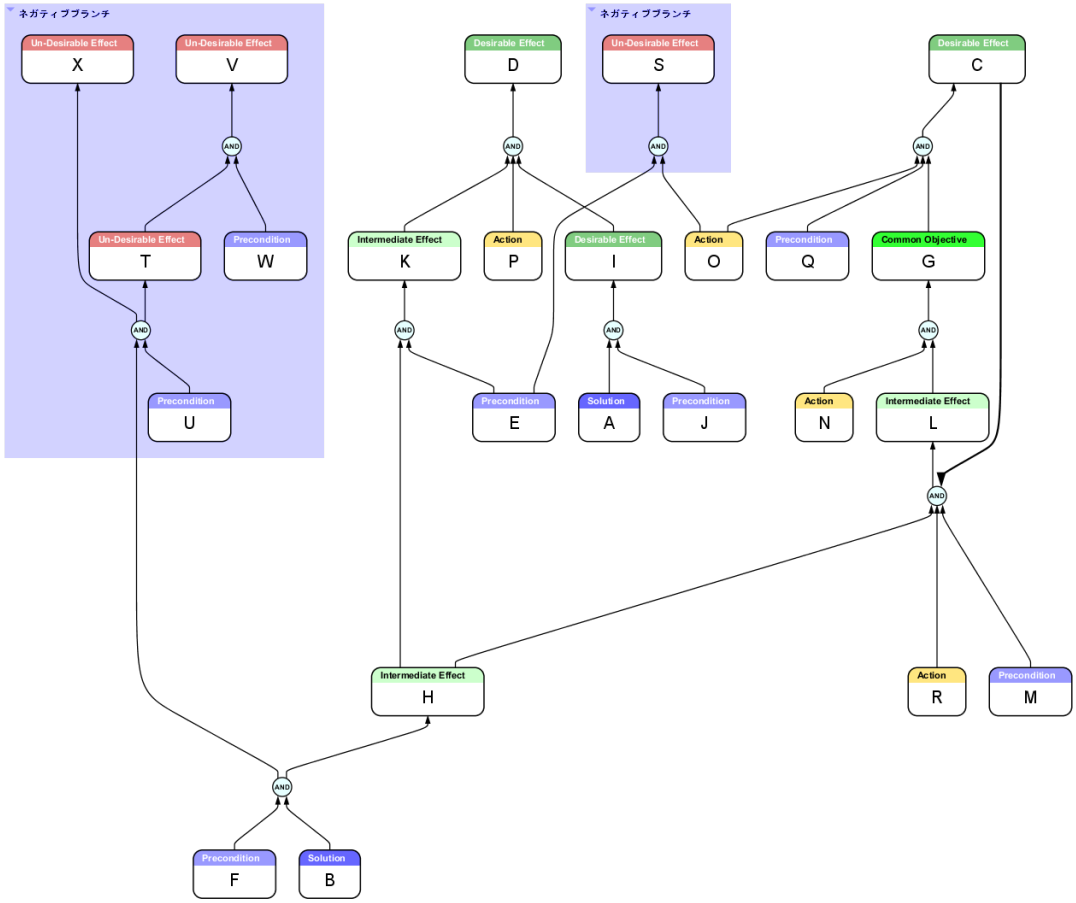
下位の結果を強化する好ましい結果を探す。このような場合があれば、後ろ向きの辺を用いて注釈をつける。ポジティブな強化ループの存在に十分な原因を作るために付加的な Action を追加する必要があるかもしれない箇所に細心の注意を払う。



ステップ 6：ネガティブブランチを探して解決する

これは重要なステップである。FRT に好ましくない結果を追加したかどうかを調べ直すべきときであり、追加したエンティティの結果である他の UDE を注意深く探す。

それが終わったら、因果のつながりで UDE が現れる最初の場所を探す。これらの「転換点」は**ネガティブブランチ**の始まりである。解決しようとする問題よりも悪い問題が起きるのを避けるには、ネガティブブランチの解決が重要である。



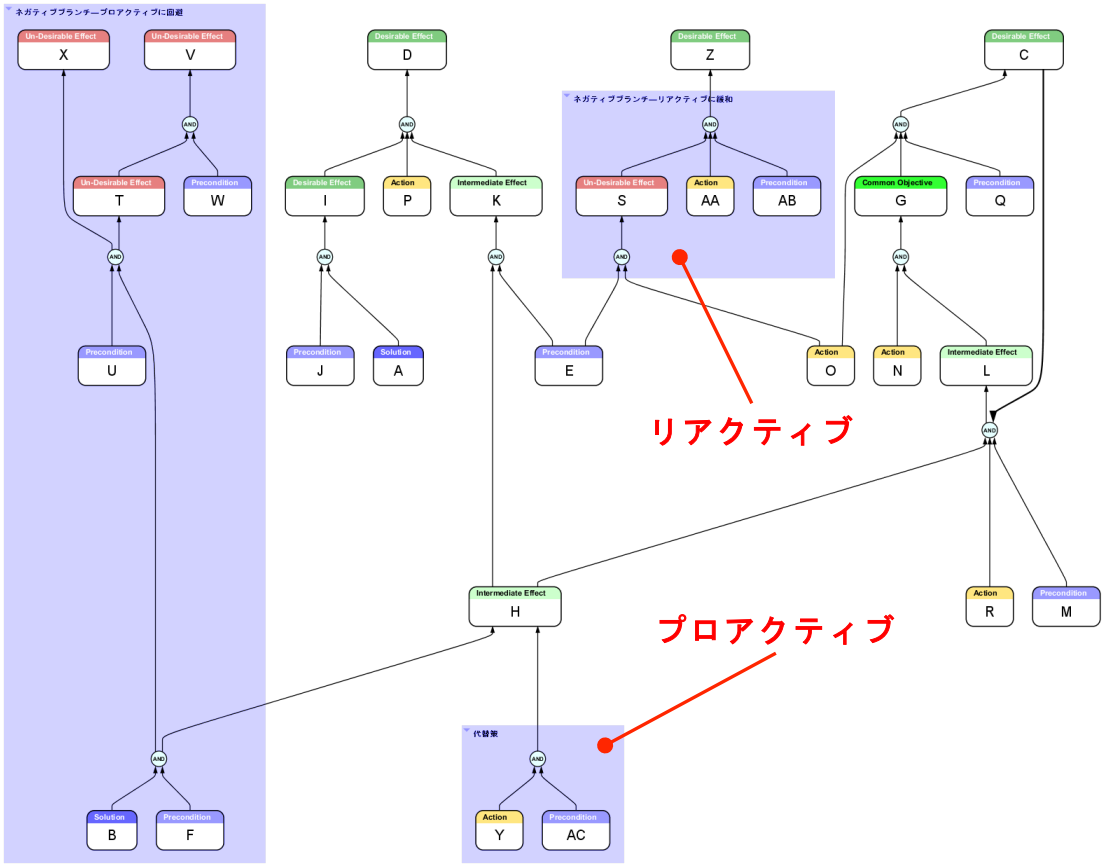
ネガティブブランチを解決する手法は 2 つある。**リアクティブ**と**プロアクティブ**である。

リアクティブな手法では、UDE の出現は許される（おそらく予期さえされている）が、避けられないと思われている。UDE を緩和する他の結果を起こすために（必要に応じて他のエンティティとともに）新しい Action エンティティ（注入）が UDE と組み合わせられる。引き起こす結果は Desirable Effect が望ましいが、中立的な Intermediate Effect であってもよい。

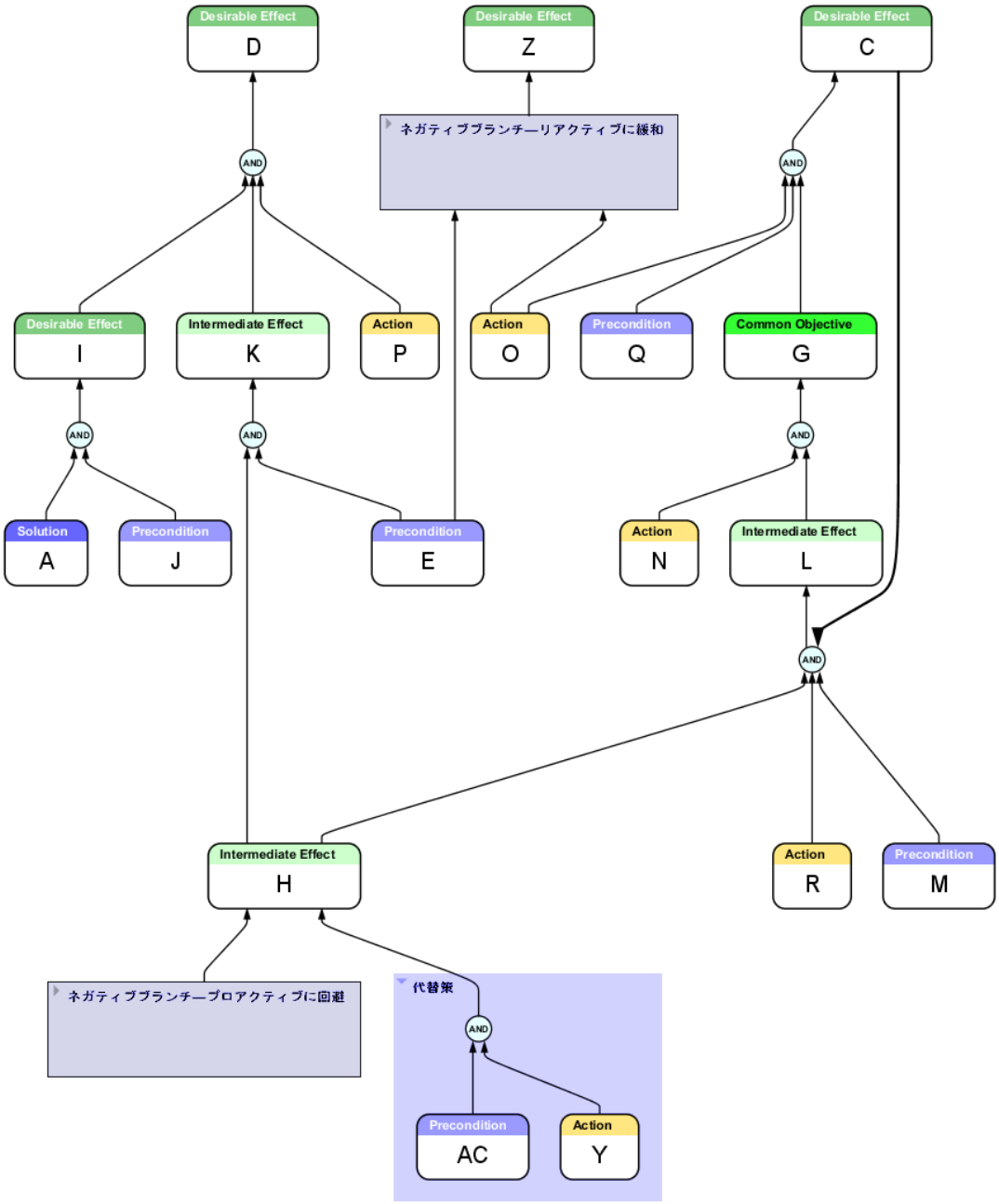
プロアクティブな手法では、最終的な Desirable Effect に至る経路上に、Intermediate Effect の次の段階を実現する新たな注入が作成され、UDE が生じることは**ない**。これは「ネガティブブランチの刈り込み」とも言われる。

以下の図では、もとの Solution エンティティ **B** を放棄し、新しい行動指針 **Y** を考案することにより、1 つのネガティブブランチを解決する。もう 1 つのネガティブブランチは、UDE **S** が発生した場合に緩和する新しい Action **AA** を考案することにより、**リ**

アクティブに扱われる。



よい経路を作成したら、ネガティブブランチに含まれるエンティティを削除する代わりに、折り畳んだグループに入れて記録しておくとよい。



誰かが善意でした提案に懸念があるときは、考える時間をくれるよう頼み、その提案を受けて、提案を初期の注入として最下部に置き、提案者が予測した Desirable Effect と、読者が予見する UDE を最終的な結果として含む FRT を構築するのがよいプラクティスである。FRT ができたら、提案者と一緒に詳細を議論できる。読者または提案者が UDE を解決する注入を作成できたら、おそらく読者が受け入れられる提案となっている。

前提ツリー（PRT）

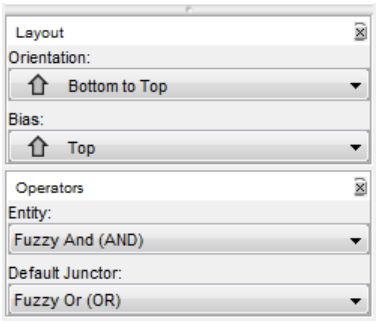
おそらく現在ツリー（CRT）を用いて中核ドライバーを描き終えたことだろう。雲を用いて有望な解決策を得て、未来ツリー（FRT）を用いて実行可能と考える解決策を立案したかもしれない。しかし、状況が極めて単純でない限り、まだやり終えてはいない。計画で最も見逃されることの 1 つが、**必要**なのに**やり終えていない**ことの特定である。これらは、ゴールへの途中で立ちはだかる障害である。これらの障害を克服する方法が開発されているので、さらなる障害が明らかとなることがある。前提ツリー（PRT）は、すべての障害を特定し、見通し、必要な行動すべてを計画に盛り込んだことを確認する助けとなるツールである。

Flying Logic の設定

PRT は必要条件の思考に基づいている。Flying Logic 文書はデフォルトで十分原因の思考向けに設定されているので、Operator ポップアップメニューで以下のように設定したいと思うだろう。

- Entity Operator: Fuzzy And (AND)
- Default Junctor Operator: Fuzzy Or (OR)

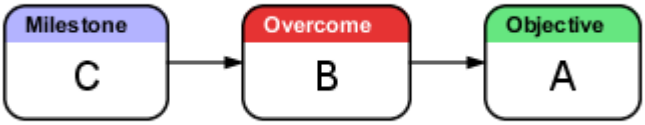
PRT は通常上から下へ、目的から始めて読む。しかし、これは辺の流れ（矢印）が目的に向かう、すなわち下から上でなければならないことを意味している。したがって、Orientation ポップアップメニューで **Bottom to Top** を指定したいだろう。



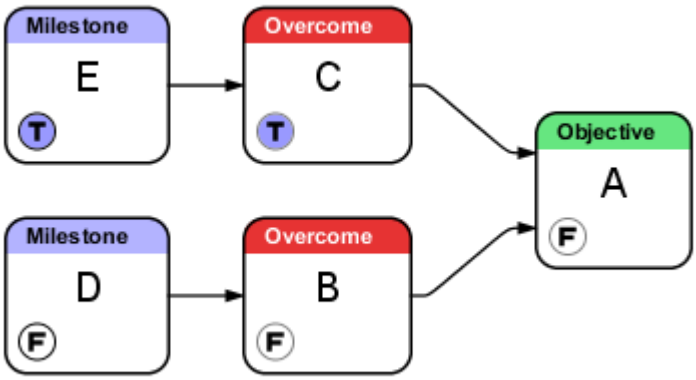
PRT は組み込みドメイン Prerequisite Tree のエンティティクラスを用いて作成し、次のクラスを用いる：Objective（目的）、Overcome（克服）、Milestone（マイルストーン）。

過去に PRT を作成した経験があれば、**Obstacle** ではなく **Overcome** を選択したことを初めは少し奇妙に感じるかもしれない。2 つの用語をある程度は交換可能として使っているが、以下の 3 つの理由で *Overcome* を用いている。

- 第一に、*Overcome* の選択により、ツリーをより自然に読めるようになる。たとえば、この単純なシーケンスは、「A を達成するには、B を克服する必要がある。B を克服するには、C を達成する必要がある」



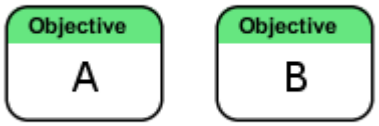
- 第 2 に、コンフィデンススピナーを用いてツリーの論理を吟味するときには、エンティティの題名にある文が存在するか現実に従っていることを示すために True を用い、現実に従っていなければ False を用いる。*Obstacle* というクラス名を用いたら、確信度の値が True ということは**障害の存在を示す**だろう。しかし、示したいことは正反対である。必要条件がすべて満たされたとき、確信度の値を True として、障害がもはや存在しない（障害が克服された）ことを示したい。したがって、Overcome エンティティを扱うときには、False が「これはまだ克服していない」（障害が存在している）、True が「これを克服した」（**障害はもはや存在しない**）ことを意味すると考えることはたやすい。したがって、PRT に含まれる全エンティティの確信度が True でないなら、何をまだ達成しないといけないのかを一見してすぐに理解できる。



- 第 3 に、これらのエンティティを *Overcome* と呼ぶ積極的な理由がある。この名前は、すべての障害は罠のもとであるという考えを分かりやすく伝えるので、計画立案者の障害に対する見方を、自分たちの**妨害をする何か**ではなく、むしろ**解決すべきもの**とすることができる。

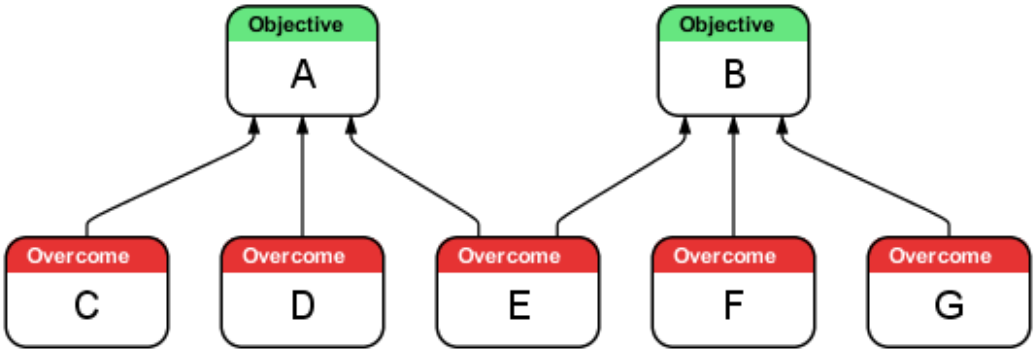
ステップ 1: 目的を特定する

Objective エンティティを作成し、現在時制の簡潔な表現で題名をつける。PRT は通常、達成しようとしている成果を定義する 1 つの Objective エンティティをもつ。しかし、緊密に関連する複数の Objective をもつこともある。Objective の表現は、未来ツリーで用いた注入（Solution エンティティまたは Action エンティティ）から得られることが多い。



ステップ 2: 克服すべき障害を特定する

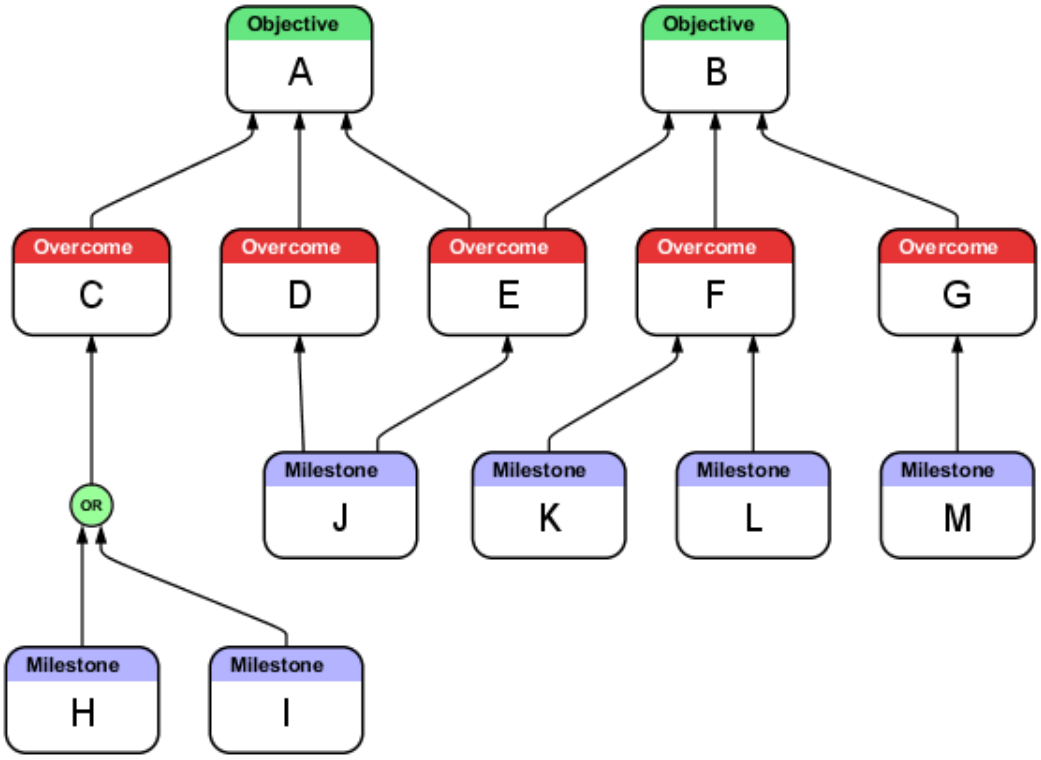
目的を達成する障害になるものがある。あるいは、すでにそれを解決したかもしれない。Objective を達成するための**存在していない**必要条件を表す、一連の Overcome エンティティを作成する。要点は、Objective を達成するのに必要なものすべてを挙げるのではなく、**まだ欠けているもの**を特定することである。各 Overcome エンティティを Objective の子孫としてつなぐ。



ステップ 3: マイルストーンをブレーンストームする

追加した Overcome エンティティをそれぞれ検討し、障害を否定する 1 つ以上の Milestone エンティティをブレーンストームする。障害を無効にするのに完全に破壊する必要はないことを記憶に留めておこう。たとえば、(比喩的には) 回り道しても、乗り越えても、下を通ってもよい。要点は、創造的であることである。

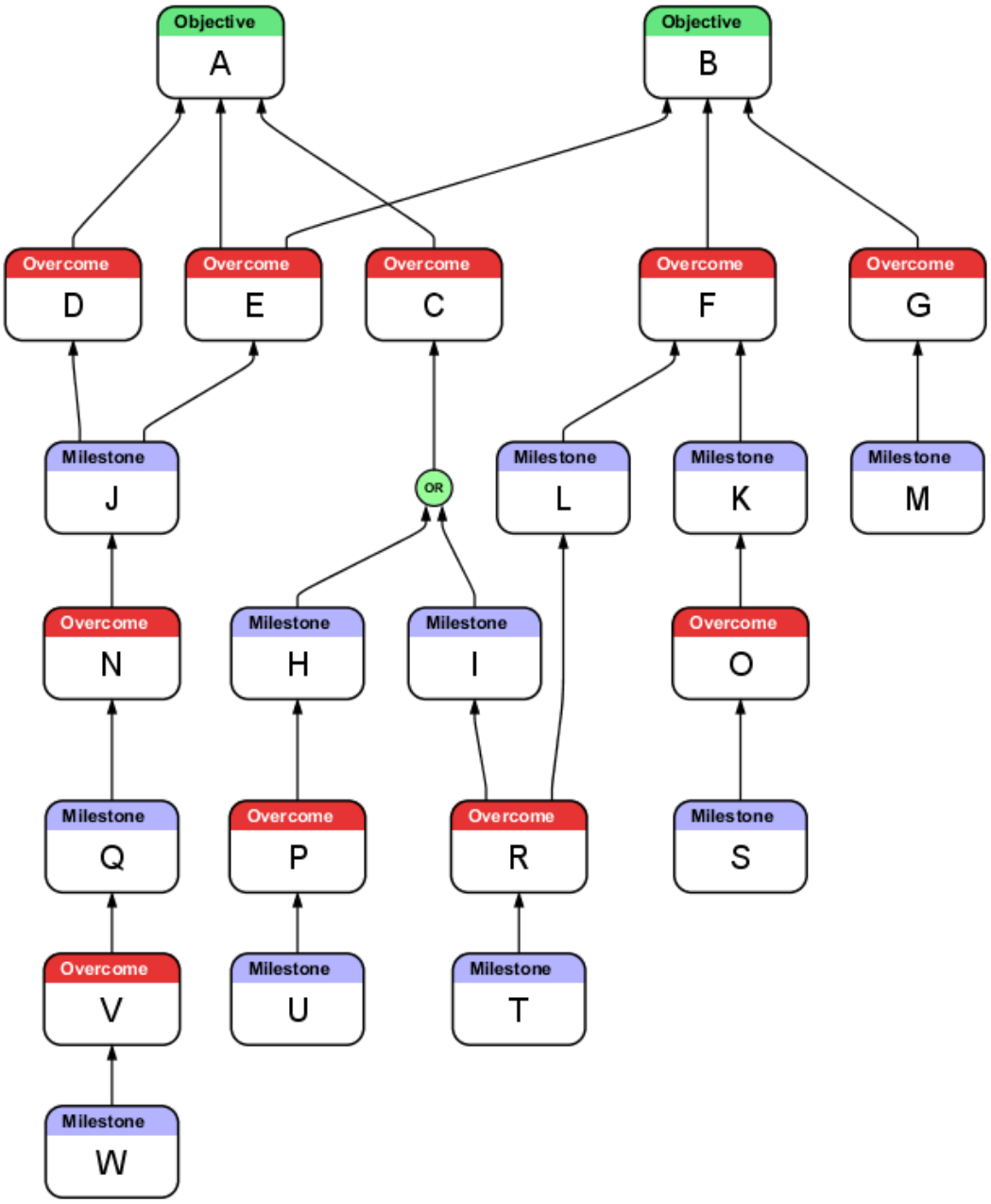
- 各 Overcome に 1 つの Milestone が対応することがよくある。(**M** は **G** の克服に必要である。)
- 特定した Milestone は複数の Overcome を克服することがある。(**J** は **D** と **E** の克服に必要である。)
- ブレーンストーミングの結果、1 つの Overcome を克服できる可能性のある選択肢が複数得られることがある。これをモデル化するには、OR ジャンクターを使用できる。ジャンクターは、既存のエンティティを線へとドラッグすることで簡単に作成できる。(**H** または **I** が **C** を克服するのに必要である。)
- 1 つの Overcome を克服するために複数の Milestone が必要となることもある。(**K** と **L** が **F** の克服に必要である。)



ステップ 4：ツリーの深掘りを続ける

直前のステップで追加した Milestone をそれぞれ検討する。Milestone を実現する際、どんな障害が現れるか。知識の欠如？ 人手不足？ 資金不足？ PRT の作成では、現在欠けている、これらの必要条件を見つけることに焦点を当てる。つまり、これが障害の本当の姿である。特定した各障害に対して新しい Overcome エンティティを作成し、適切な Milestone の先祖として接続する。新しく追加された各 Overcome エンティティに対して、それを解決する Milestone を考案する。等々。

ツリーを深掘りするに従い、追加する Milestone は、より小さく扱いやすくなる。あるところで、実現に大きな障害が見あたらない Milestone を追加することになる。これらの Milestone が PRT の根であり、最初に取りかかって達成すべきことを示している。もちろん、Milestone を実現するには多くの実際の *Action* が必要かもしれない。これは、次章の移行ツリー（TRT）の主題である。しかしここでは、障害を生じない Milestone を特定すれば十分である。



ステップ 5: ツリーを読み上げて検証する

最も単純な Milestone から最終的な Objective に至るまでツリーがうまく接続できていると感じられたら、明快さと完全性のためにツリーを注意深く読み上げるべきときである。PRT は必要条件の思考を用いて作成されているので、ツリーは Objective から始めて辺の流れと逆向きに読む必要がある。ツリーの各段階は、以下のパターンの 1 つで読む。

- **A** を達成するためには、**B** を克服しなければならない。
- **B** を克服するには、**C** を達成しなければならない。

ツリーを読み上げるときに、妥当性基準を適用していることを確認する。以下のように自問する。

- 本当に、この障害を克服する必要があるか？
- この障害を克服する必要を避ける方法はあるか？
- このマイルストーンは本当にこの障害を克服するか？
- この障害を克服するのに他に必要なマイルストーンはあるか？
- この障害を克服するのに十分なマイルストーンが他にもあるか？

この段階で、Flying Logic のコンフィデンススピナーを使って、根の Milestone から Objective に至るまでの辺の流れをステップごとに追って図を検査するとよい。

ステップ 6：ツリーを刈り込んで仕上げる

PRT が論理的に健全であることを検証したあと、選択可能な 2 つ以上の Milestone（OR 関係で接続されている）を含んでいるかもしれない。棄却した選択肢は、削除するか折り畳んだグループに置いて、PRT を刈り込む。

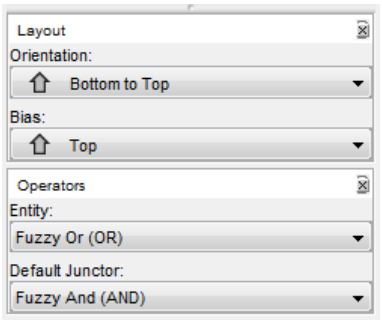
移行ツリー（TRT）

ゴールの達成を阻害する障害を特定し、それらを克服するマイルストーンを作成したら、今度は実行計画が必要となる。システムが確実にゴールへと近づく段階を形成する、一連の行動である。計画を見る人（理想的には立案に参加）が、各行動、とくに自分に役割のある行動が、システム全体にどのように貢献するかを、明確に分かるようにすべきである。これが**賛同を得る**鍵である。つまり未来像の共有が熱心な協力を生み出す。移行ツリー（TRT）は、**現在ツリー**（CRT）から**未来ツリー**（FRT）への移行を実現する実行計画の作成に有効なツールである。

移行ツリーは、一連の行動を含んでいるという点で、プロジェクト管理で用いられる伝統的な PERT 図と類似しているが、両者の主な違いの 1 つは、TRT が各行動と対になっている前提条件（実状に関する仮定）を含んでいることである。このことは、TRT が、計画実行時点の Precondition を誘因とする、いくつかの緊急対応計画を含むことを意味している。本質的には、計画の実行が進展するに従い、「現場の」状況に応じていくつかの異なる PERT 図が TRT から「発生する」ことがある。これによって、TRT は、高いリスクを含む計画を作成するのに理想的なツールとなっている。

Flying Logic の設定

TRT は十分条件の思考に基づいており、これが Flying Logic 文書が最初に作成されたときの設定なので、TRT の作成を始めるときに Operators ポップアップメニューで何か特別なことをする必要はない。TRT は、ゴールを上部に置いて、上へと流れることが多い。したがって、文書の向きを変更するため、Orientation ポップアップメニューを **Bottom to Top** に設定したいかもしれない。



TRT は、組み込みドメイン Effects-Based Planning のエンティティクラスを使って作成する。基本的には次のクラスを用いる：Goal、Precondition、Intermediate Effect、Action。計画の便益を強調するために Desirable Effect を用いることもできる。また、

一連の行動によって、今後の計画実行で解決しなければならない不可避な UDE が生じるなら、Un-Desirable Effect エンティティを用いてもよい。

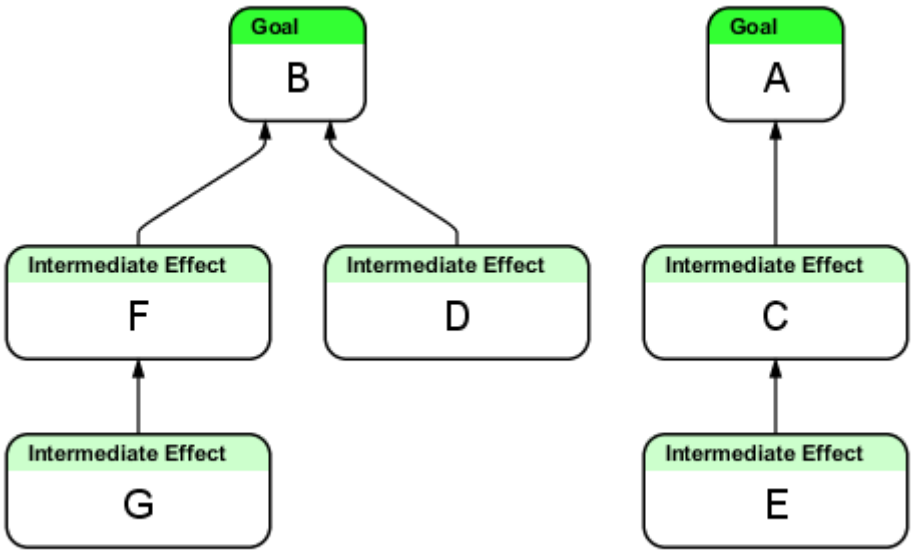
ステップ 1:ゴールを特定する

TRT は 1 つの Goal エンティティをもつことが多いが、合理的に関連しあうならば複数の Goal があってもよい。Goal はかくあるべきという直観から TRT を始めることができ、また未来ツリーの注入の 1 つまたは前提条件の Objective から作成した Goal から始めることもできる。いずれの場合であっても、Goal エンティティは、望まれる実状を示す明快な現在時制の文を題名としなければならない。

ステップ 2:中間結果を特定する

前提ツリー（PRT）を作成したなら、移行ツリー（TRT）文書へ直接コピーすることのできる Milestone エンティティの組が存在する。しかし、PRT の Milestone はすべて**必要**だが、おそらく十分でないということに気づくことが重要である。PRT は**まだ行っていないこと**を特定して克服するために用いられるが、TRT は行うべきこと**すべて**と、それを実行する順序を特定するために用いられる。

PRT から Milestone をコピーしないのなら、ゴールに達成するのに必要だとわかっている状態を表す Intermediate Effect を作成して辺で接続し、ある程度順序を定義したいかもしれない。この段階では、絶対的な厳格さは必要ない。正確な因果関係の順序を定義するのは、以下のステップの焦点である。

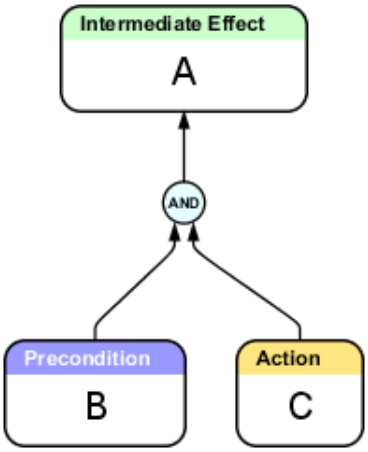


ステップ 3:完全なステップを定義する

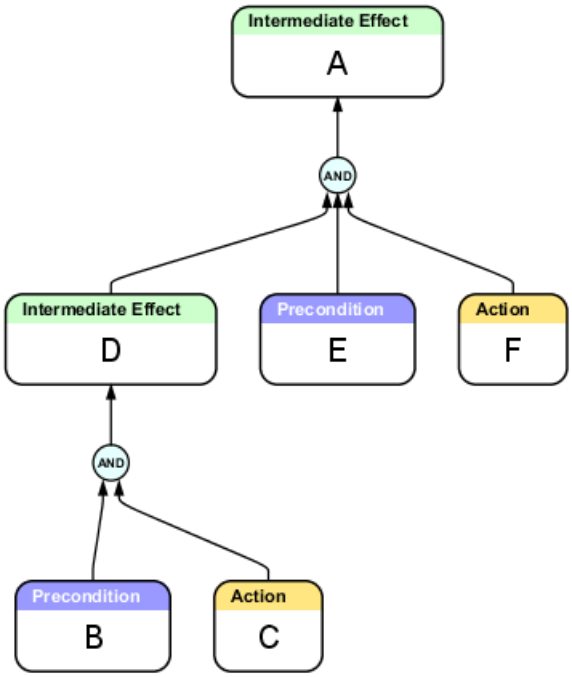
実行計画の**ステップ**は 3 つのことを必要とする。

- 達成しようとしている成果。これは、Intermediate Effect、PRT からコピーした Milestone、TRT の Goal のいずれかである。
- 現在ツリーの文。これは Precondition エンティティ(統制範囲外の現実の一面を表し、それゆえに与件だと捉えなければならない)、もしくは前のステップの成果である Intermediate Effect または Milestone である。
- Action。明確であるためには、Action は統制または影響の範囲内にあり、実行が成功したことを判断する明確な基準をもち、権限と実行能力とともに資源に割り当てられるものでなければならない。

現在の実状と行動は、成果を達成するための必要条件および十分原因として論理的に組み合わせられなければならない。



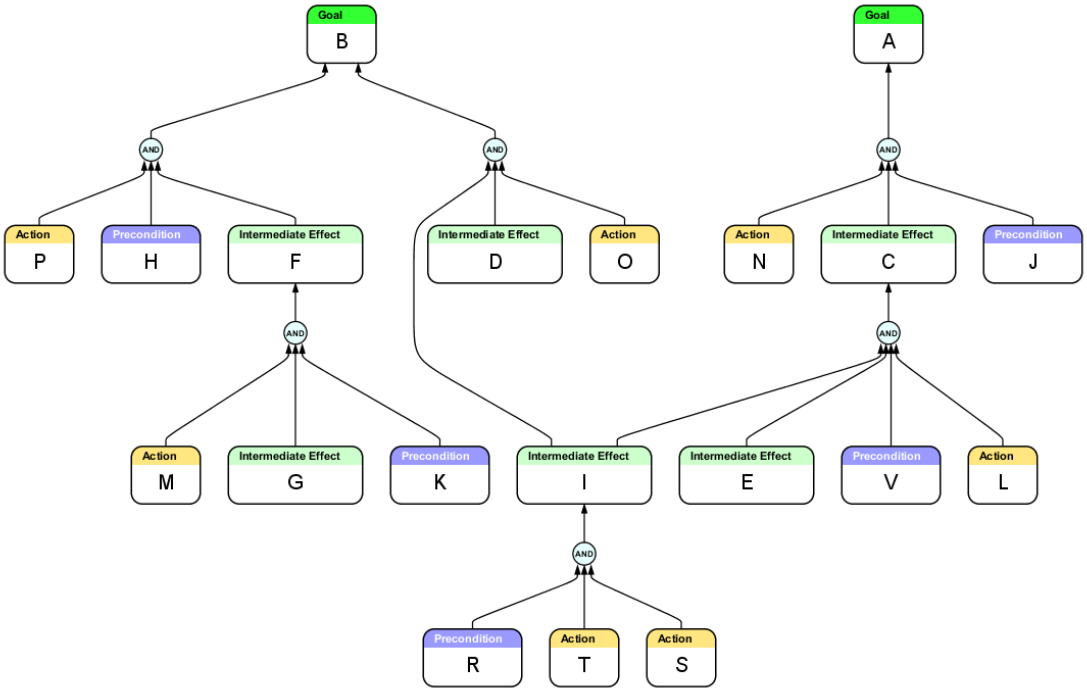
ステップを読んでいるときに、行動が結果を生じるのに十分でなければ、それは 1 つ以上の十分原因のサブステップに分解する必要がある。



ステップ 4:ツリーの構築を続ける

TRT の各中間結果は、完全な**ステップ**と同様に記述されなければならない。行動の結果と現在の実状である。これらのステップは直線的なシーケンスとなることが多いが、

並行したシーケンスとなるか、より複雑な依存性をもつこともある。



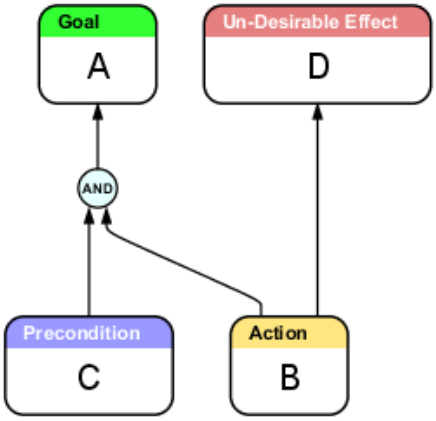
ステップ 5: 副作用を探して解決する

これは、未来ツリー（FRT）の記述における同名のステップと同様である。実際、TRT は、注入から始めて結果に至る代わりに、望まれる結果（ゴール）から始めて、それを達成する注入に至る作業を行う、ある種の FRT と考えることもできる。

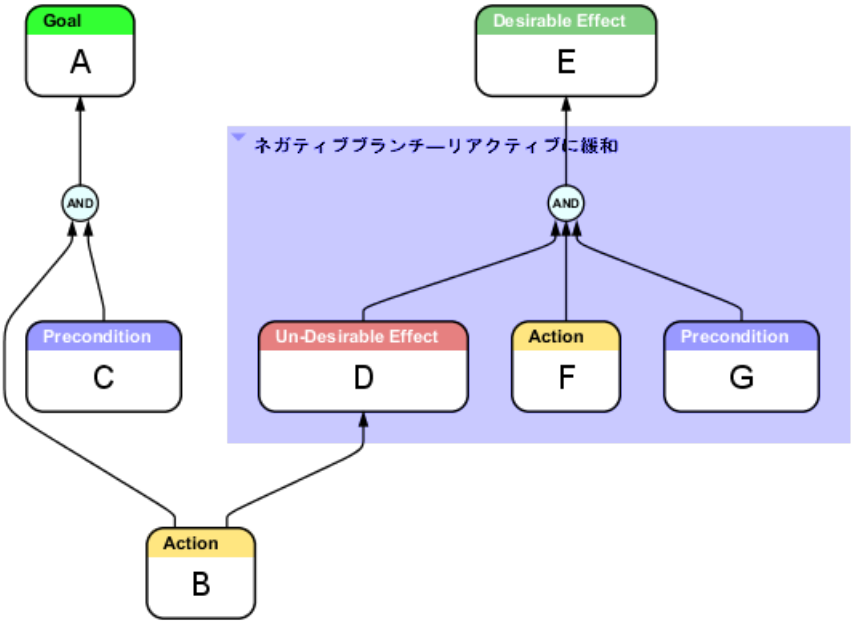
すでに作成した FRT から作業を行うときには、行動はすでに、副作用の特徴である望ましくない結果（UDE）を避けるように考えられているかもしれない。

他方、リスクを回避できないこともある。リスクは、行動を起こすときに、その前提（実状に関する仮定）が **True** にならなかったときに発生する。計画を実行する環境の特性に依存して、計画の実行時点で**どの** Precondition が真になるかを正確に予測することは非常に困難であり、実状を硬直的に捉えた計画を作成したのであれば、実状が合わなかったときに失望する可能性が高いだろう。したがって、計画が孕むリスクの度合いには、Precondition が成立しなかったときに発生する UDE を特定し、それらの UDE を緩和する（リアクティブな対策）か、それらを避ける（プロアクティブな対策）か、いずれかの代替案を用意することがきわめて重要である。

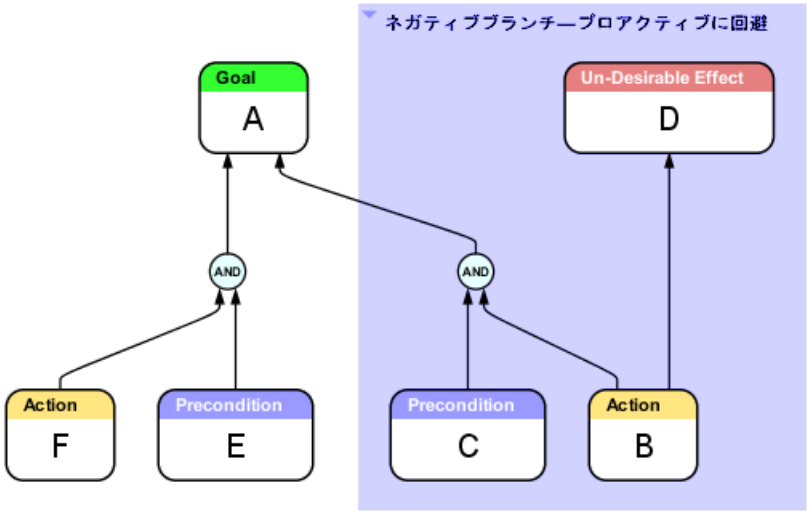
計画が解決しない UDE で終わっているなら、それは未完成である。



完成した計画は、Goal または Desirable Effect のみで終わる。



完成した計画：リアクティブな緩和策



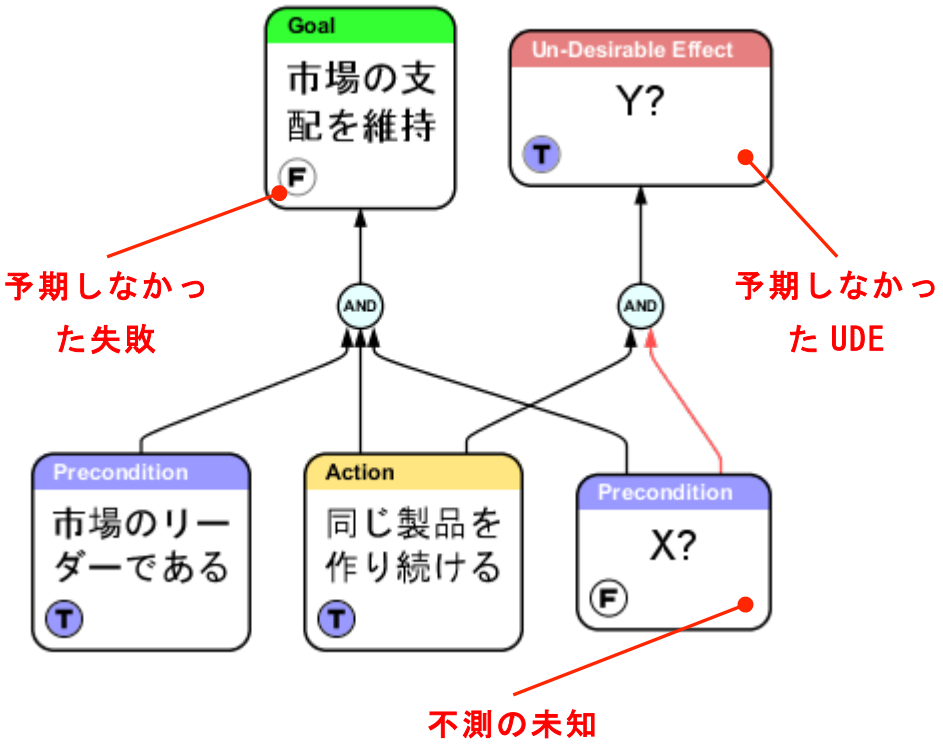
完成した計画：プロアクティブな回避策

最後に、ある特定の結果に対する必要条件および十分条件と判断されたものすべてが存在するが、計画が実行されたときに結果が生じないとしたら、何が起きるだろうか。予想しなかった UDE が現れたら何が起きるだろうか。さらに重要なことには、この悪夢が発生する機会を減少させるためにどうしたらよいだろうか。

これは、**予期しなかった不確実性**（**不測の未知**とも言う）が発生した場合である。想像あるいは予期しなかった、そしてし得なかった事柄である。この場合、事前に決定した緊急時対応策は存在し得ないが、準備できることはいくつかある。

- 可能であれば、いくつかの解決策を並行して検討し、最終的に最良の策に取り組む。
- 傲りを避ける。謙虚な組織文化を育み、自らの専門性によって盲目になることを防ぐ。
- 変化する状況に対して、柔軟であり、計画を適応させることを望む。
- 懐疑が（その時点では）明確に述べられなかったとしても、経験豊富な利害関係者の直感と懸念を真剣に受け止める。

最後の場合については、不明瞭な懐疑であっても、決定していない Precondition として TRT に追加し、具体化できなかったときには後から削除することができる。不測の未知を、可能な場所すべてに追加してはならない。強力だが具体的でない懐疑が表明された場所に限る。不測の未知は、計画の実行時に計画した結果が生じなかった場合に、評価の一部として追加することもできる。



ステップ 6:ツリーを読み上げ、検証する

これは、未来ツリー（FRT）の記述における同名のステップと同様である。緊急時対応計画を含めた場合は、ツリーの関連する部分を 2 回以上たどり、各回でツリーの異なる経路をたどるよう Precondition を設定する。

戦略&戦術ツリー（S&T ツリー）

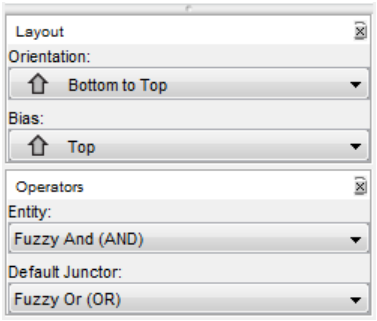
最新の TOC-TP ツールである戦略&戦術ツリー（S&T ツリー）は、最上位の組織目標から、包括的かつ多層的で、十分に検証された実現ステップへと詳細化するために用いられる。これは、組織の各部が演じる役割を明確化することで、大きな組織全体の広範な改善を実現するのに用いられる。

Flying Logic の準備

S&T ツリーは、必要条件の思考に基づいている。Flying Logic 文書はデフォルトで十分原因の思考向けに設定されているので、Operator ポップアップメニューで以下のように設定したいと思うだろう。

- Entity Operator: Fuzzy And (AND)
- Default Junctor Operator: Fuzzy Or (OR)

S&T ツリーは通常上から下へ、最上位の戦略を出発点として読む。しかし、これは辺の流れ（矢印）が目的に向かう、すなわち下から上でなければならないことを意味している。したがって、Orientation ポップアップメニューで **Bottom to Top** を指定したいだろう。

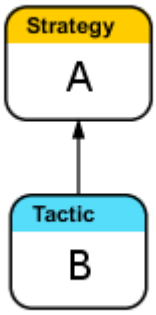


S&T ツリーは、**Examples/Strategy & Tactics Tree** フォルダのドメインファイル **Strategy & Tactics Tree.logic-d** が提供するエンティティクラスを用いて作成する。**Entity ➧ Import Domain** コマンドを用いて既存の文書へインポートすることもできるし、**File ➧ Open** コマンドを使って開くこともできる。後者ではテンプレート文書のように振る舞い、S&T Tree ドメインが読み込まれて準備のできた、無題の新しい文書が開く。

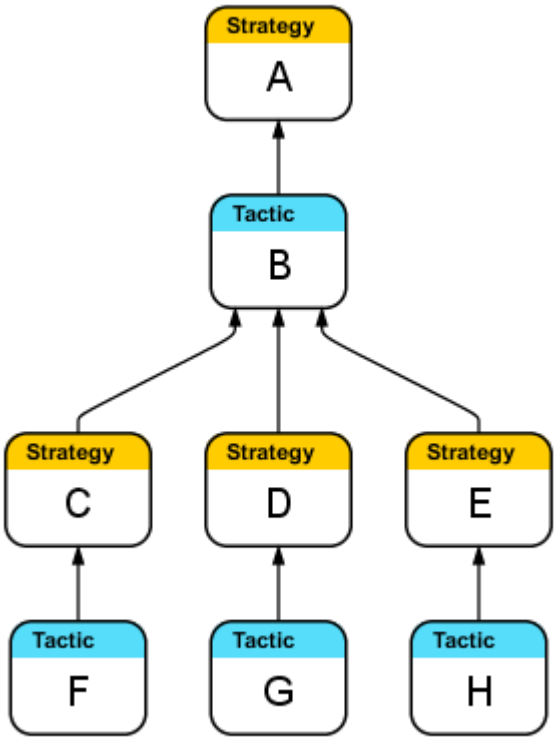
S&T ツリーの構造

S&T ツリーは、戦略と戦術がツリーのような階層をもつ行動を記述する相補的な概念だという考えに基づいている。ツリーの各ステップ（ノード）は自身の存在を戦略とともに示す。つまり、ノードが存在する理由の記述である。最上位の戦略は、システムのゴールに対応する。

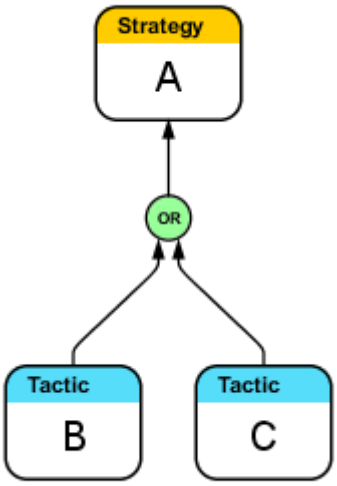
各 **Strategy** は、その戦略がどう実現されるかを記述する 1 つの **Tactic** エンティティによって支持される。S&T ツリーの底辺は、常に Tactic の層である。それは、戦略を支持する最も基本的な行動である。



1 つの Strategy を実現するのに 2 つ以上の Tactic が必要となる場合、1 つの Tactic が 2 つ以上の副 Tactic に分解されるかもしれないが、それぞれはまず、その戦略によって正当化されなくてはならない。したがって、各 Strategy は正確に 1 つの Tactic を直下にもつが、Tactic は副 Strategy を**もたない**か、**2 つ以上もつ**場合もある。

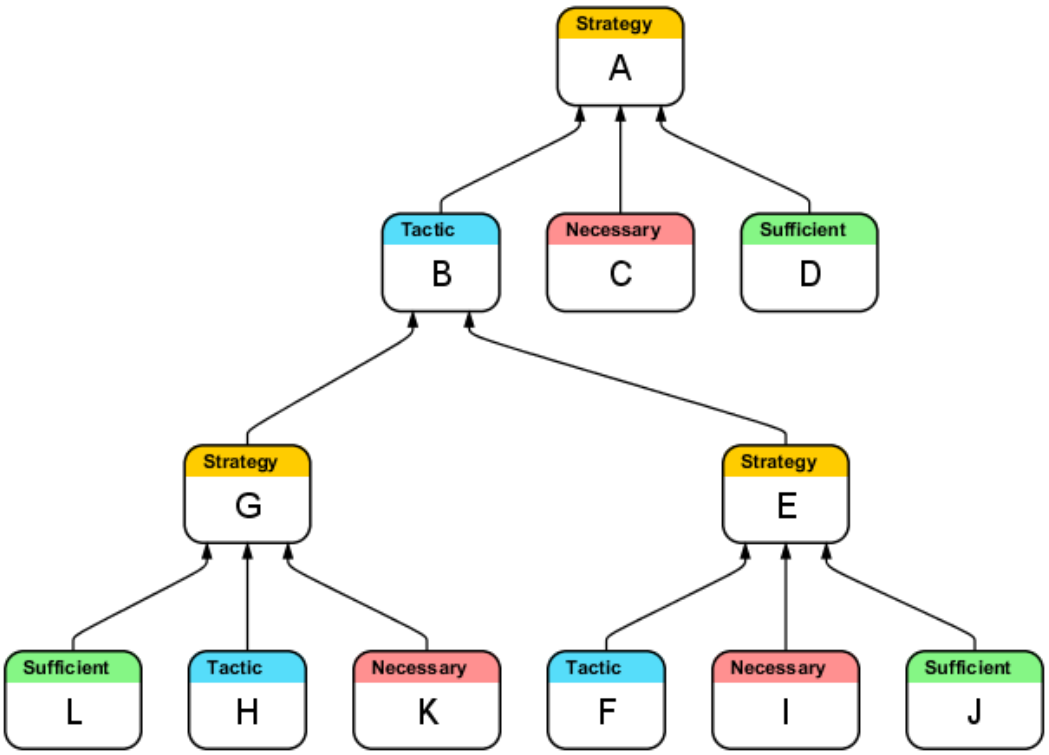


Strategy がそれを達成する可能性のある Tactic を 2 つ以上もつ場合には、それらを OR 関係で結ぶ。



Strategy を達成するには、Tactic を提供するだけでは十分でない。その Tactic が親の戦略を達成するために必要かつ十分であることを示す必要がある。したがって、Necessary エンティティと Sufficient エンティティを作成し、Tactic エンティティの兄

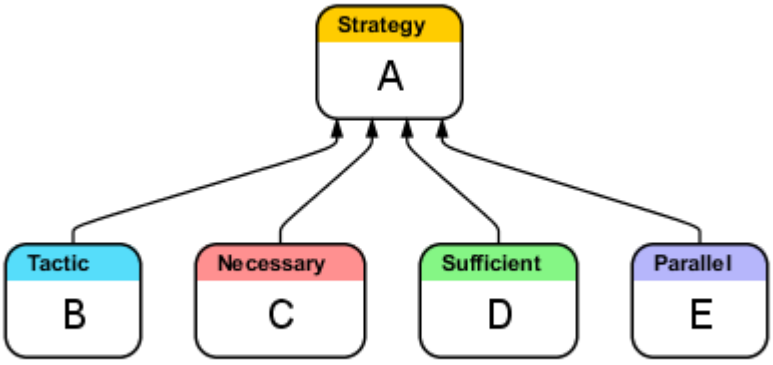
弟とする必要がある。各エンティティに与えられる題名は、クラス名が示す事柄を正確に行わなければならない。Tactic が絶対に機能する理由（十分性）とともに、戦略を達成するために Tactic が実現されなければならない理由（必要性）を記述する。Tactic が必要または十分な根拠が複数ある場合には、追加の **Necessary** または **Sufficient** エンティティを追加することもできるし、そのエンティティのテキスト注釈に列挙することもできる。



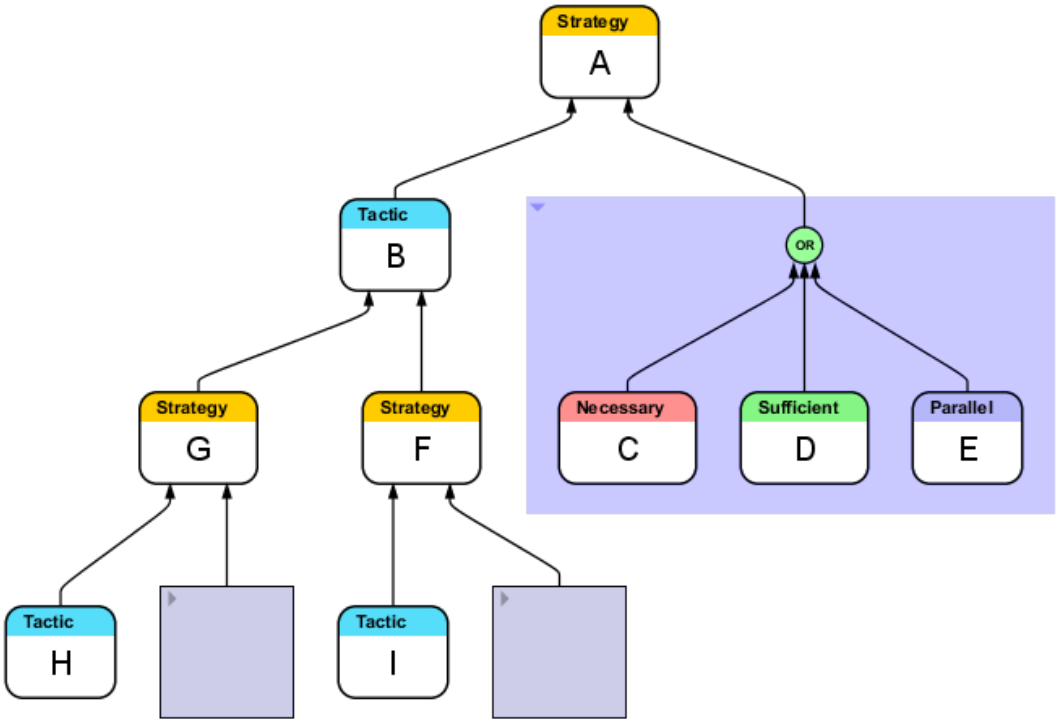
もう 1 つのエンティティクラス **Parallel**（並行する仮定）を、以下のような、Tactics の必要性あるいは十分性に直接言及しない反論に対して回答するために用いることができる。

- すでに Strategy が存在している。それを実現するために取るべき行動はない。
- Tactic を実現することはできない。

これらを組み合わせ、5 種のエンティティすべてで**ステップ**を構成する。

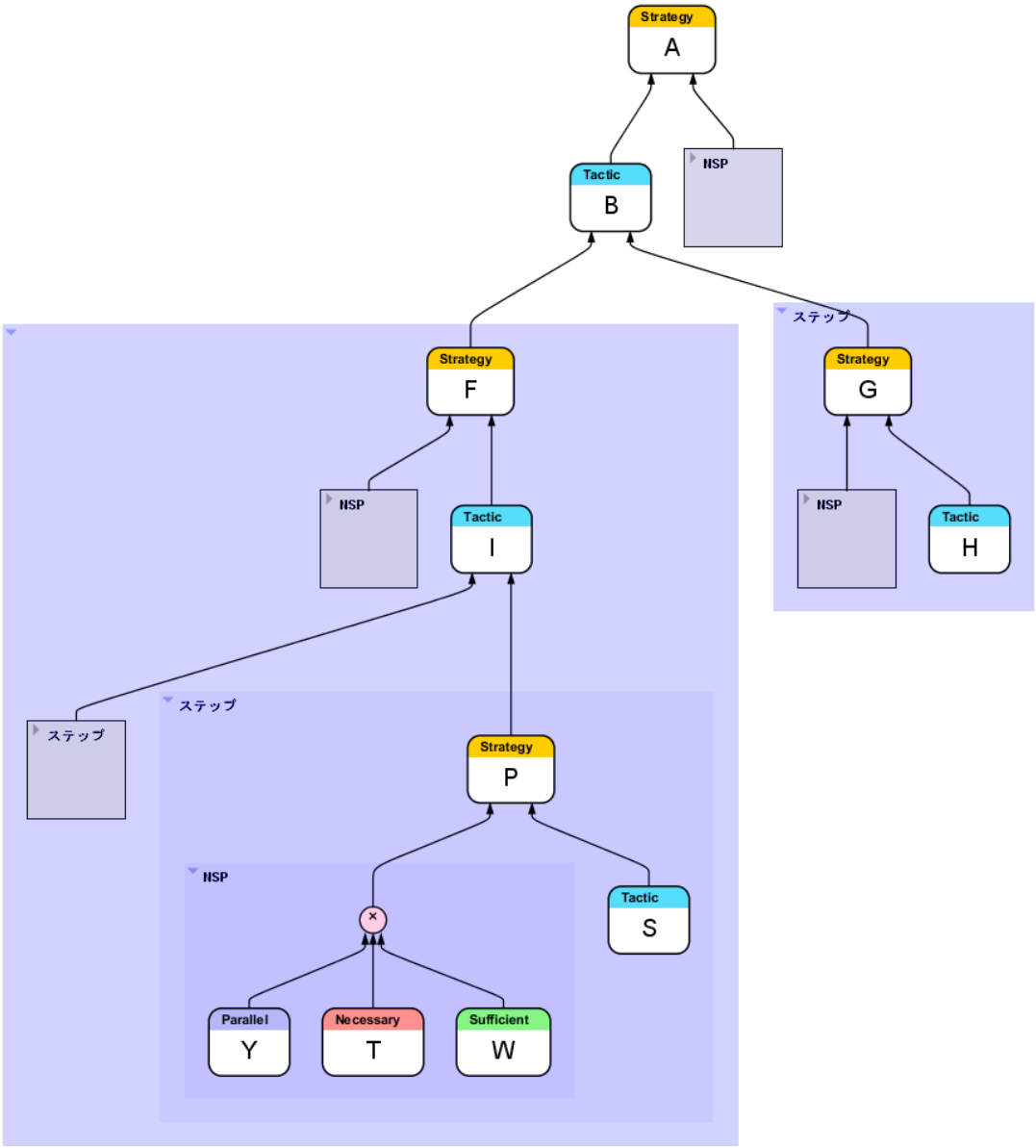


S&T ツリーは非常に大きくなることがあるので、Flying Logic のグループ化機能が図の管理に役立つ。ひとつの方法は、Strategy を支援するエンティティすべてをグループ化し、AND ジャンクターで結合して、グループから出る辺を 1 つだけにすることである。



グループはさらに、Strategy エンティティ、Tactic エンティティ、および Necessity・Sufficient・Parallel（NSP）エンティティからなるサブグループを含む、全体のステップをグループ化するのに用いることができる。この技法を用いると、非常に大きな S&T ツリーを整えて、必要な水準の詳細へと「ドリルダウン」することが容易にでき

る。これらのアイディアは提案と捉え、大きな Flying Logic の図を管理するための自分なりの技法を自由に開発してよい。



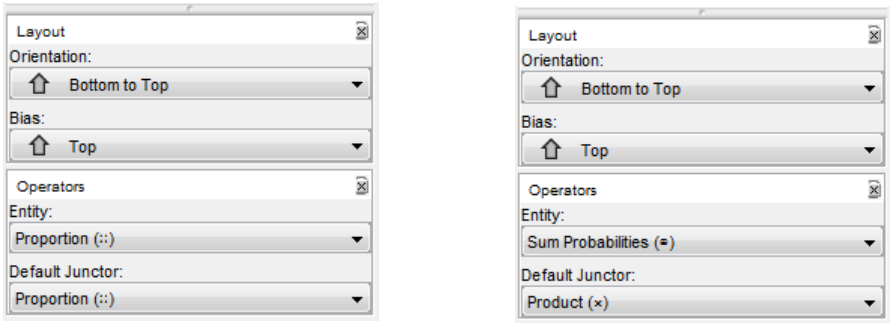
第３部　他の技法

証拠に基づく分析（Evidence-Based Analysis）

このカテゴリーのエンティティクラスは、より確率的なモードでの分析が望まれる環境に適している。証拠に基づく分析が有用な実世界のシナリオの 1 つは、競合分析である。通常、分析が行われ、ある期間それが実行される。その間に、成立する命題が発見されるかもしれない。それは、分析を行う団体の更なる行動の契機になるかもしれない。

Flying Logic の準備

証拠に基づく分析には、2 つのスタイルがある。**信念ネットワークスタイル**と**確率的スタイル**である。信念ネットワークスタイルが用いられる場合、Flying Logic 文書のエンティティ演算子とデフォルトジャンクター演算子はともに Proportional（比例、::）を設定する。確率スタイルが選択された場合、典型的には、エンティティ演算子は Sum Probabilities（確率の和、⊕）、デフォルトジャンクター演算子は Product（論理積、×）である。この設定は、十分原因の思考における Fuzzy Or（OR）および Fuzzy And（AND）の用法に類似している。



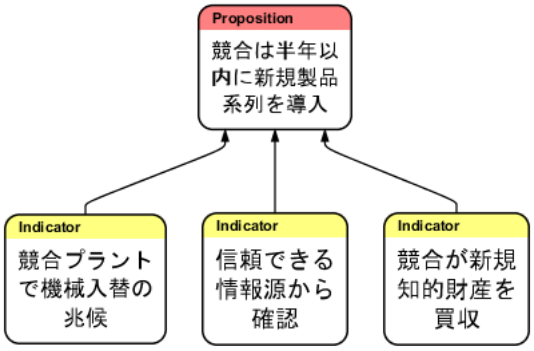
信念ネットワーク/確率モデルの設定

命題（Proposition）

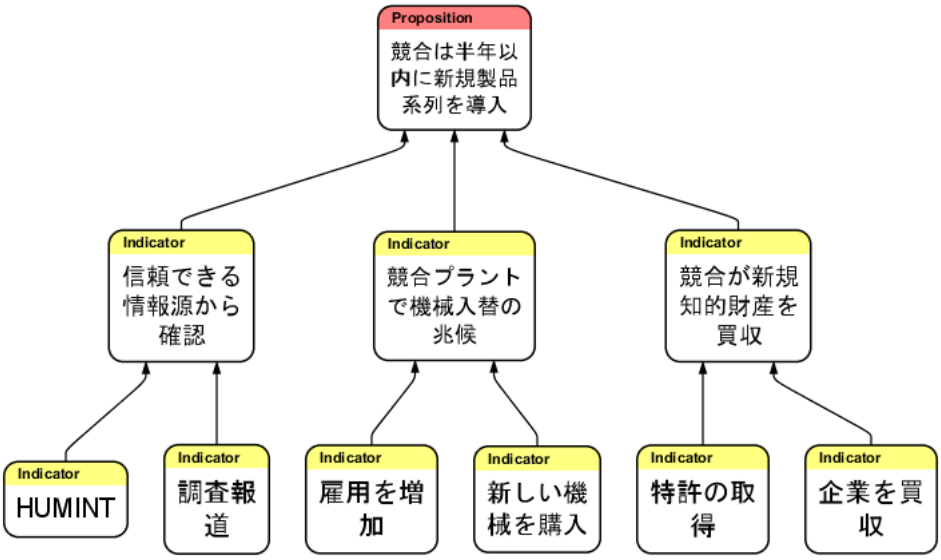
命題（要件とも言う）は、分析によって最もありそうな回答を見つけたい問いである。命題は真となる確率がある文の形を取る。命題の確率が定められた閾値を越えるかどうかが、証拠に基づく分析の主要な目的の 1 つである。命題は、Effects-Based Planning におけるゴールに類似しており、従って終端となる。すなわち、それらは常に子孫であり、決して先祖にはならない。

指標（Indicator）

指標は命題または他の指標の潜在的な原因であり、したがって Effects-Based Planning の Intermediate Effects に類似していると考えられる。指標の他の考え方は、推論された証拠である。各命題は、典型的には、そこにつながる指標の集合をもつ。各指標は命題の原因のひとつと考えられ、総体としてその命題が成立すること（すなわち、**要件が満たされている**こと）を認識するための「テンプレート」を形成する。



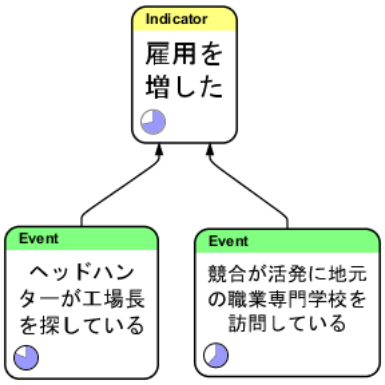
各指標は次に、そこにつながり、検討中の指標がおそらく成立するということを認識する「サブテンプレート」を形成する、より具体的な指標の集合をもつことがある。指標は、子孫でも先祖でもあることが普通である。



複雑な分析においては、特定の指標に対して個々の分析者が割り当てられることがある。分析者は、直接責任をもつ指標の先祖となる全指標に対して監督的な役割に置かれる。

事象（Event）

事象は、分析のライフサイクルを通じて知るところとなった、直接的な証拠である。インテリジェンス（諜報）のコミュニティを例に取ると、事象は無線インテリジェンス（SIGINT）、人的インテリジェンス（HUMINT）、公開情報インテリジェンス（OSINT）などから得られる。事象は常に先祖であり、決して子孫にはならない。それらには、**信頼性**（あるいは確率）に基づいて確信度が割り当てられる。

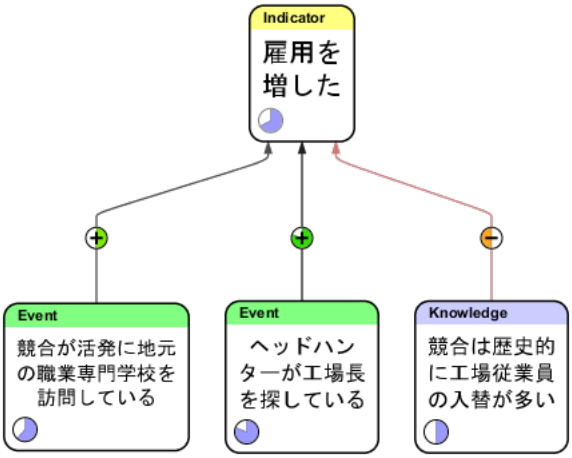


ナレッジ

ナレッジは、分析対象の状況に関して正しいことが分かっている、適切な事実を表す。ナレッジは、事象が通知される前に分析に組み入れることも、事象の発生時に分析に加えることもできる。Knowledge エンティティは、Event との組み合わせで、それとつながる様々な指標を支持あるいは否定する文脈と意味論を提供する。Event と同様に、Knowledge エンティティは先祖であって子孫ではない。それらには、信頼性（あるいは確率）に基づいて確信度が割り当てられる。

辺の重み

モデル中の辺の重みは、各エンティティと子孫との間での正（または負）の相関関係に基づいて割り当てられる。

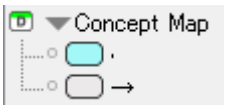


概念マップ

概念マップは、関連する概念のネットワークの速やかな理解を、可視化および獲得あるいは伝達するために用いられる。

Flying Logic の準備

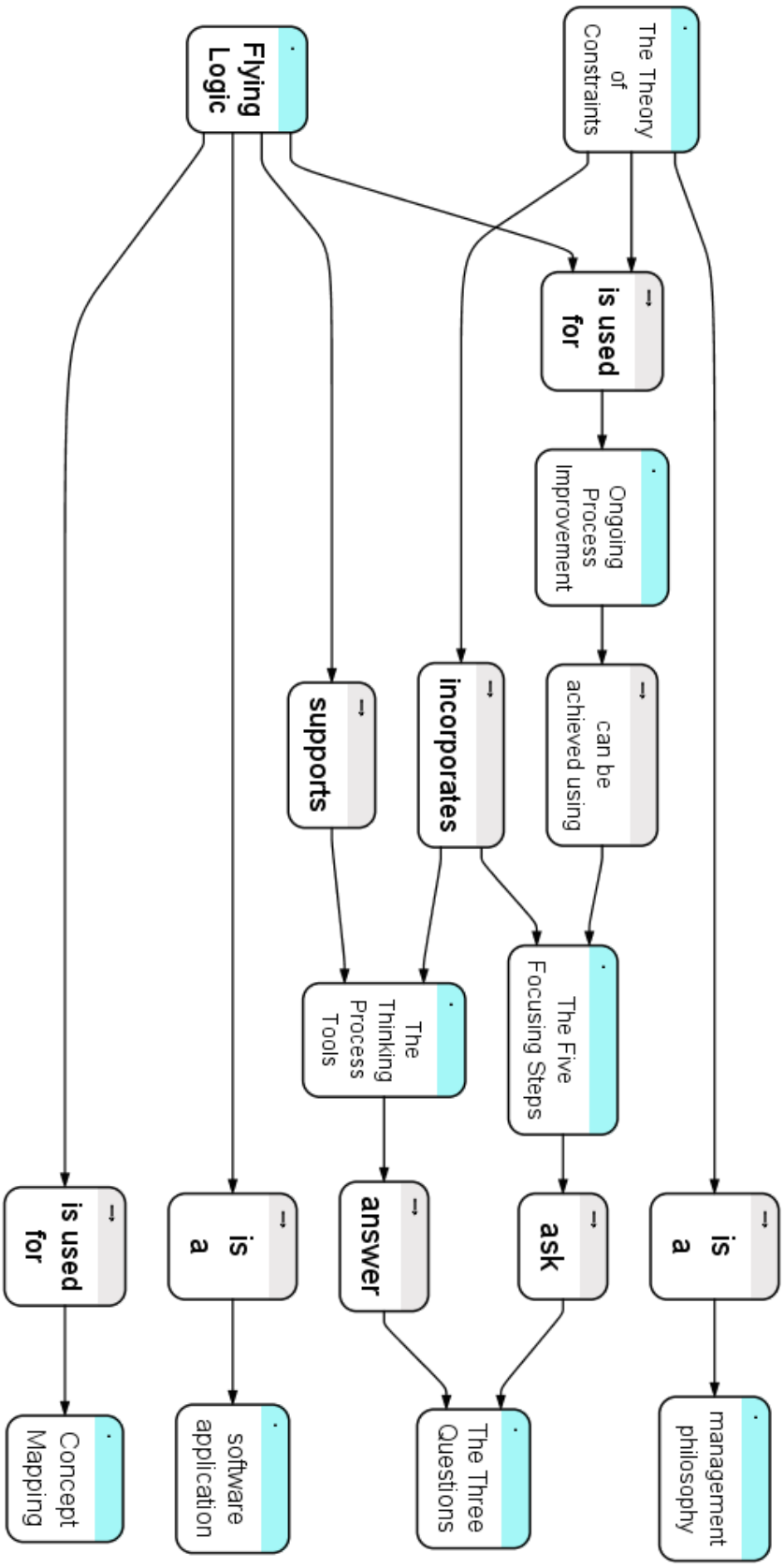
概念マップは、**Examples/Concept Maps** フォルダにあるドメインファイル **Concept Map.logic-d** で提供されるエンティティクラスを用いて作成する。このドメインを既存の文書に **Entity ‣ Import Domain** コマンドで読み込んでもよいし、**File ‣ Open** コマンドでこれを開くこともできる。この場合はテンプレート文書のように振る舞い、Concept Map ドメインが読み込まれて使う準備が整った、無題の新しい文書が開く。



概念マップの構造

概念マップでは 2 つのエンティティクラス、**Concept**（·） と **Relation**（→）を用いる。エンティティクラス名を言葉で示す代わりに記号を用いているのは、概念マップはすべてエンティティの題名で読み、「概念」や「関係」という言葉はまったく用いられないからである。

概念マップは 1 つ以上の主要な概念を根として始まり、関係は支持する概念間をつなぐために用いられる。概念マップを作成するときの主な規則は、各 Concept→Relation→Concept のステップは完全な文として読めなければならないということだ。関連する概念を任意の順序で追加でき、必要な箇所に接続することができる。


The Theory of Constraints
Flying Logic
is used for
Ongoing Process Improvement
can be achieved using
incorporates
supports
The Five Focusing Steps
The Thinking Process Tools
is a
ask
answer
is a
is used for
management philosophy
The Three Questions
software application
Concept Mapping

付録

リソース

Flying Logic Web サイト

FlyingLogic.com のリソースは、Flying Logic のユーザーに限らず、TOC、事業改善、個人的な改善に興味をもつ方々一般を意図している。

- Flying Logic フォーラム－議論の場
- Flying Logic ウィキ－協業による知識ベース
- Flying Logic ブログ－ニュースおよび関心事

TOC に関する Web サイト

- My Saga to Improve Production

  by Eli Goldratt
- TOC.tv

  Videos by Eli Goldratt
- A Guide to Implementing the Theory of Constraints

  by Kelvyn Youngman
- TOC Video Overviews

  by Dr. James R. Holt, Washington State University
- Theory of Constraints International Certification Organization — Among other things, the TOC-ICO hosts a yearly conference
- TOC Glossary

  Pinnacle Strategies
- Strategy and Tactics — a description of the S&T Tree

  by Eli Goldratt, Rami Goldratt, and Eli Abramov
- Throughput Accounting — includes a description of Throughput, Inventory, and Operating Expense

  Wikipedia

TOC に関する書籍

- The Logical Thinking Process: A Systems Approach to Complex Problem Solving

  by H. William Dettmer
- Thinking for a Change: Putting the TOC Thinking Processes to Use

  by Lisa J. Scheinkopf

  （邦訳：頭のいい人の思考プロセス―すぐに使える、図と論理の問題解決スキル、リサ・J. シェインコフ　※絶版）
- Introduction to the Theory of Constraints (TOC) Management System

  by Thomas B. McMullen, Jr.

心理学、コミュニケーション、交渉に関する書籍

- なぜあの人はあやまちを認めないのか

  エリオット・アロンソン、キャロル・タヴリス
- ハーバード流交渉術　必ず「望む結果」を引き出せる！

  ロジャー・フィッシャー、ウィリアム・ユーリー
- 【決定版】ハーバード流“NO”と言わせない交渉術

  ウィリアム・ユーリー
- 話す技術・聞く技術―交渉で最高の成果を引き出す「3 つの会話」

  ダグラス・ストーン、ブルース・パットン、シーラ・ヒーン、ロジャー・フィッシャー
- ダイアローグスマート　肝心なときに本音で話し合える対話の技術

  ケリー・パターソン他

他の有用な Web サイト

- The Theory Underlying Concept Maps and How To Construct Them

  by Joseph D. Novak, Alberto J. Cañas, Florida Institute for Human and Machine Cognition