

20120622

拡張されたストック・フローによる システム記述

- 交換代数と経済交換のシステム記述 -

社会経済システム論第九回
知能システム科学専攻 出口弘



Today's Agenda

負のストックを含むダブルエントリーのシステム状態と状態変化の記述

負のストックを含むフローの増減のタイプの分類

ダブルエントリーのシステム記述

冗長代数と交換代数

交換代数の公理系による定式化

交換代数のソフトウェア実装

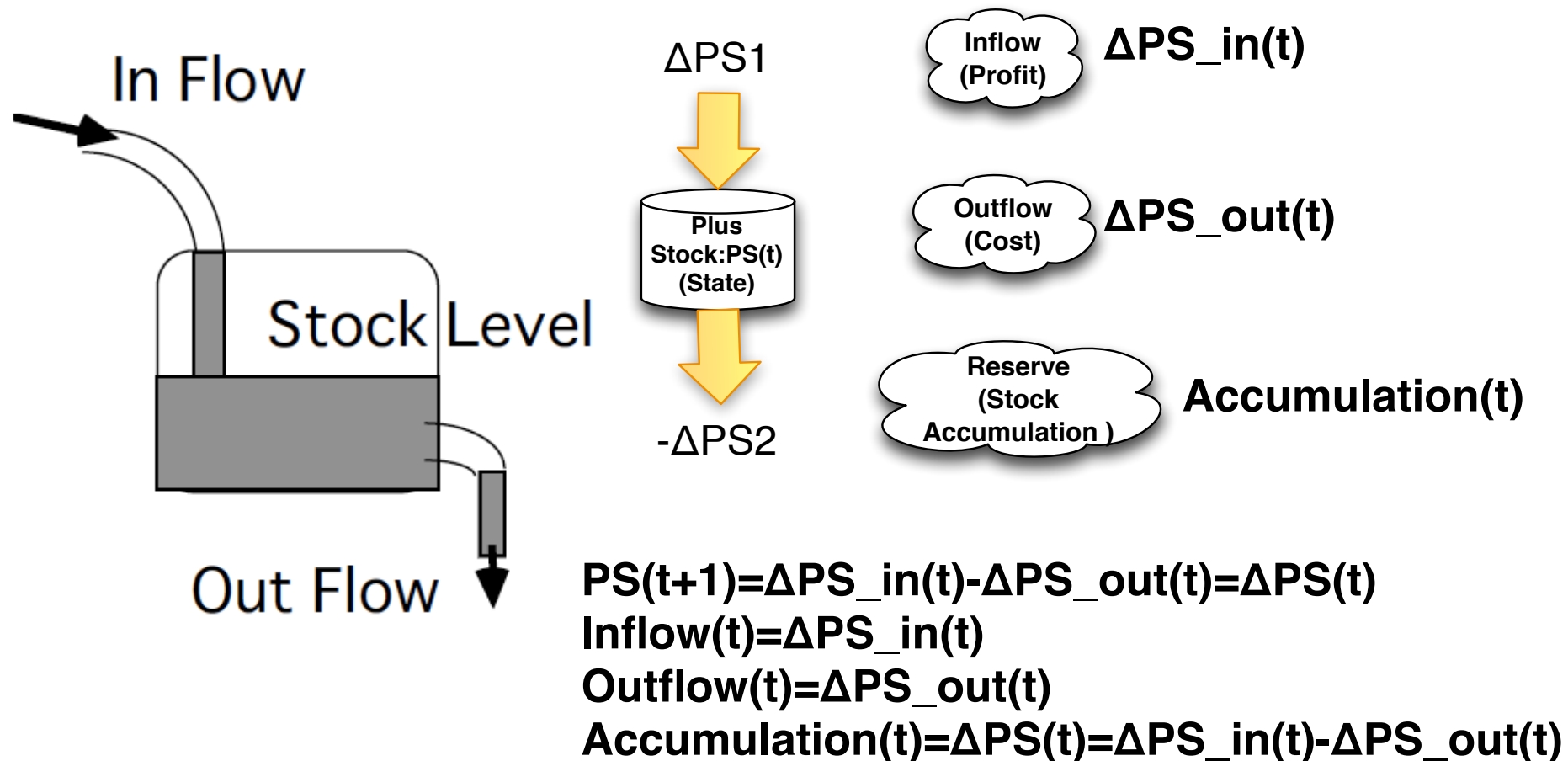
AADLの基本文法

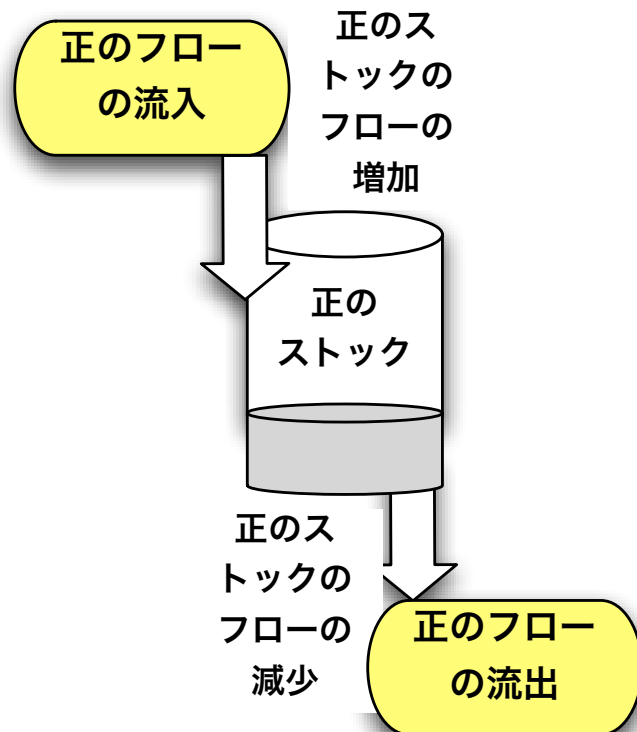


経済的イベントのシステム記述

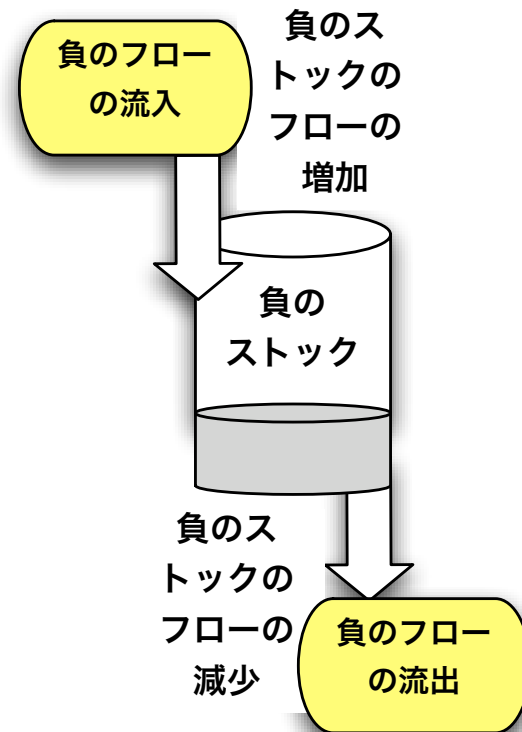
ストック・フロー型の単純なシステム記述
からより複雑なシステム記述へ

単純なシステム記述はプラスのストックとその流入流出でシステムの状態（ストック）と状態変化（フロー）は記述される。





物理学のような自然科学では何らかの物理量（変数）の状態（ストック）とその変化（フローの流入＝ストックの増加、とフローの流出＝ストックの減少）でシステムの状態変化を記述する。その変数が複数あればベクトル表示となり、流入、流出はプラスマイナスで示される。



社会のシステム記述には負債という負のストック概念によるシステム記述がどうしても必要。負債（負のストック）を状態として扱い、その変化（負のフローの流入＝負のストックの増加、と負のフローの流出＝負のストックの減少）でシステムの状態変化を記述する。負のストック概念が入る事で通常のプラスマイナスの数をつかうのではない記述が必要となる（簿記にマイナスの数が出てこないように）

ストック・フロー型のシステム 記述の拡張としての、負のス トックを含むダブルエントリー 型システム記述

**正と負のストックにより記述される経
済の状態とその変化の記述には、ス
トックフローの記述概念の拡張による
ダブルエントリー型記述が必要！！**

経済的なイベントの記述の特色

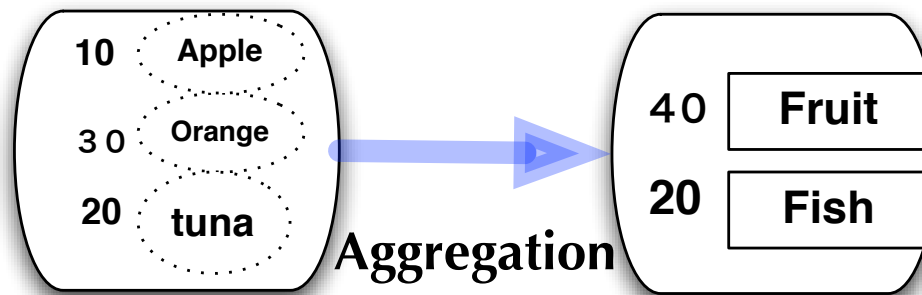
- 経済システムでのイベント記述の三つの特色
 - 1) 常に二重に分類された形で（ダブルエントリー）、何らかのイベント（フローの発生）は把握される。
 - 2) マイナスの数を用いなくて双対基底でマイナスの数を表現する（冗長代数という特色）
 - 3) マイナスのストックという特殊な項目がある。
- 経済学的なイベント記述の例：
 - リンゴを現金30円で買った：リンゴが増えて現金が減るイベント x_1 を
 - $x_1 = 30 \langle \text{リンゴ、円} \rangle + 30 \hat{\langle \text{現金、円} \rangle}$ で表現する。
 - ここで $\langle \text{リンゴ、円} \rangle$ は、分類基底と計測単位を表す複合基底であり、30はその単位で計測した数量。 $\hat{\langle \text{現金、円} \rangle}$ は現金が出て行くことを表す。現金も財の分類の単位。
 -

社会経済のデータ表現

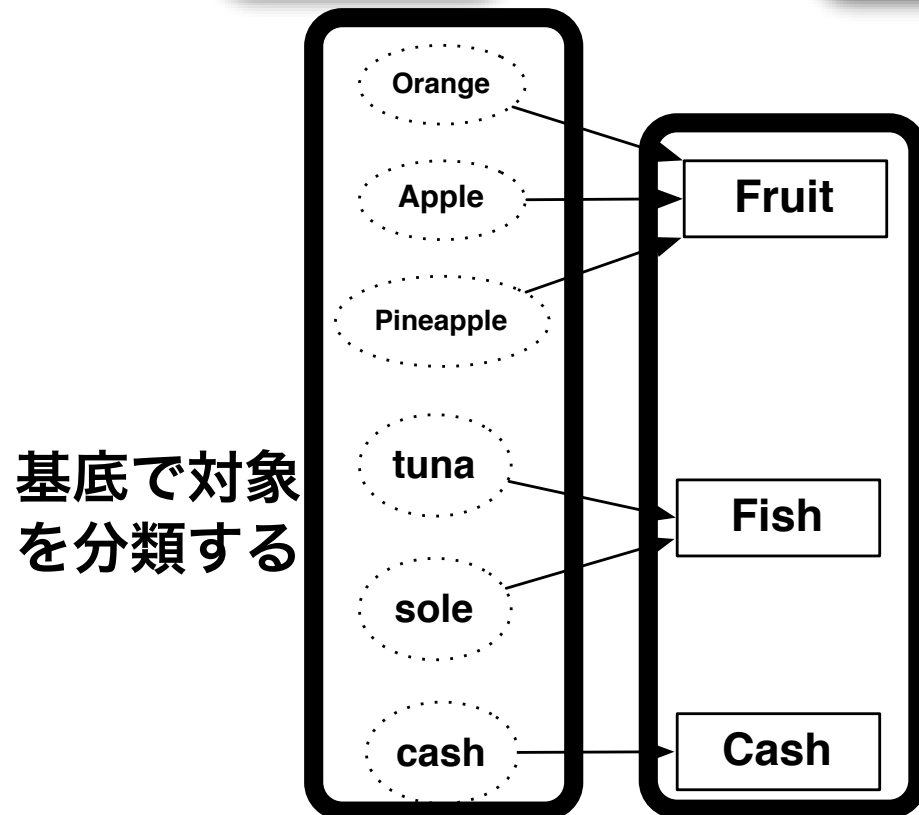
- 社会経済で状態やその変化を表すデータを表現するには
 - 1) 対象の分類を与える事
 - 2) 負債のような負のストックを扱える事
 - 3) ダブルエントリー：同時に交換関係にある二つのイベントが生じること
- の記述が必要、2) 3) は会計的な記述。

(1) 対象にある粒度で分類
を与えること：基底の設定

代数的基底を用いた財の分類と数値表現



対象を <apple>
のように代数的
な基底で分類
し、基底の値に
この分類での量
を記述する財の
ストックやフ
ローのデータ表
現法は極めて一
般的



基底間の
関係づけ
を与える
ことで、
分類替え
を行う
(この
ケースは
アグリ
ゲーション)

$\Lambda_1 = \{\text{orange, apple, pineapple, tuna, sole, cash}\}$

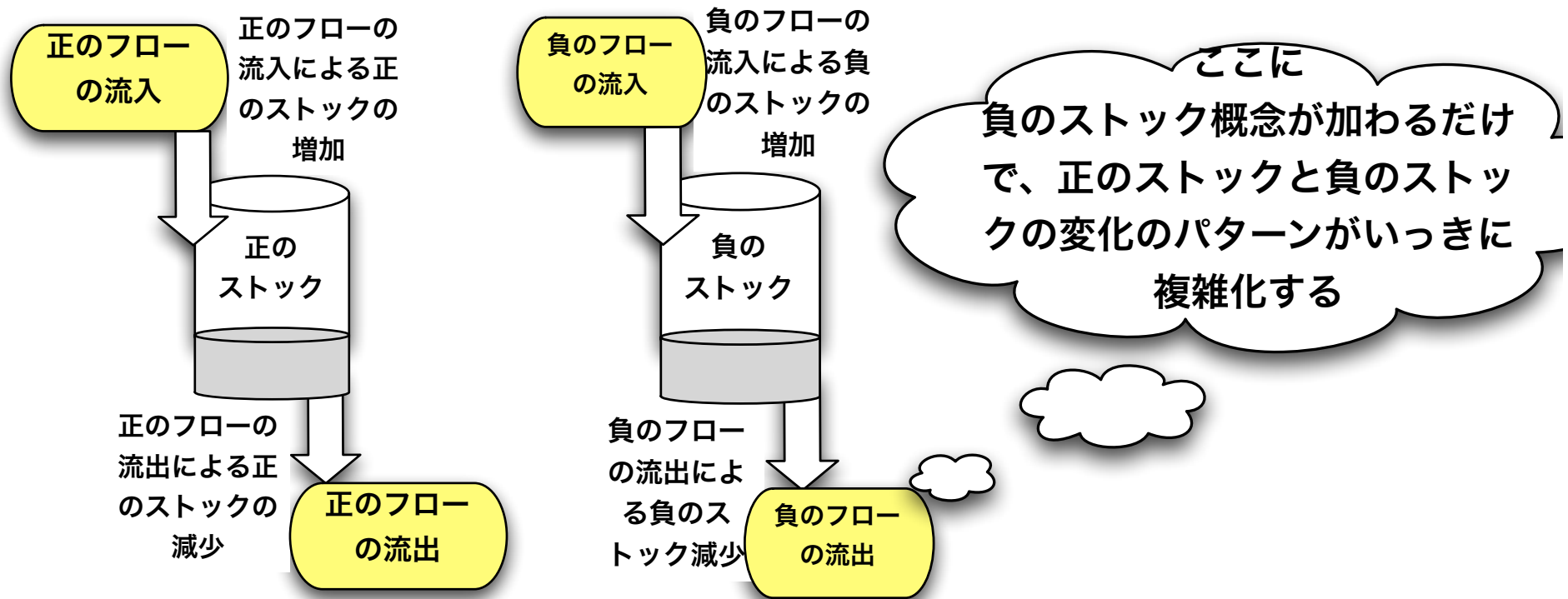
$\Lambda_2 = \{\text{fruit, fish, cash}\}$

交換（冗長）代数でのデータの表現

- 冗長代数は、何らかの分類された項目毎に与えられた数値データをまとめて表現し、その上の演算を行うための代数系である。
- 基底（分類項目）は<名前、単位、時間、主体>と4つの項目からなる。
 - 単純化する場合は、<名前、単位、#, #> (Name, Unit, Time, Subject) のように示す。
 - データは基底に対する値の組の和で表される。
 - 例： $x = 200<\text{リンゴ}, \text{価額}, \#, \#> + 400<\text{さんま}, \text{価額}, \#, \#> + \dots$
 - 例： $y = 200<\text{リンゴ}, \text{価額}, 2006\text{Q1}, \#> + 400<\text{リンゴ}, \text{価額}, 2006\text{Q2}, \#> + 720<\text{リンゴ}, \text{価額}, 2006\text{Q3}, \#> + 300<\text{リンゴ}, \text{価額}, 2006\text{Q4}, \#> + \dots$ 時系列データの表現
 - 例： $z = 200<\text{リンゴ}, \text{価額}, 2006\text{Q1}, \#> + 400<\text{リンゴ}, \text{価額}, 2006\text{Q2}, \#> + 720<\text{リンゴ}, \text{価額}, 2006\text{Q3}, \#> + 300<\text{リンゴ}, \text{価額}, 2006\text{Q4}, \#> + 400<\text{さんま}, \text{価額}, 2006\text{Q1}, \#> + 330<\text{さんま}, \text{価額}, 2006\text{Q2}, \#> + \dots$ 品名(Name)と時間(Time)に関してデータを表現したもの。

負のストックを扱える事

- 1) 負のフローとストックを扱うためのベクトル空間の拡張が冗長代数。
- 2) 冗長代数では、マイナスの数を使わず、代わりに分類基底に $\hat{}$ (ハット) を付けたハット基底と、通常基底との相殺演算 \sim (バー) というのを導入する。



＊ 社会のシステム記述には負債という負のストック概念によるシステム記述がどうしても必要。

＊ 負債（負のストック）を状態として扱い、その変化（負のフローの流入＝負のストックの増加、と負のフローの流出＝負のストックの減少）でシステムの状態変化を記述する。

＊ 負のストック概念が入る事で通常のプラスマイナスの数を使わないシステムの記述が必要となる（簿記にマイナスの数が出てこないように）

ダブルエントリー記述

1) 交換で変化するシステムは、二つのイベントの同時生起という形でシステムの変化が記述される。これをダブルエントリー（複式）記述という。これによりあるイベントが何のイベントかと言うことを記述できる。

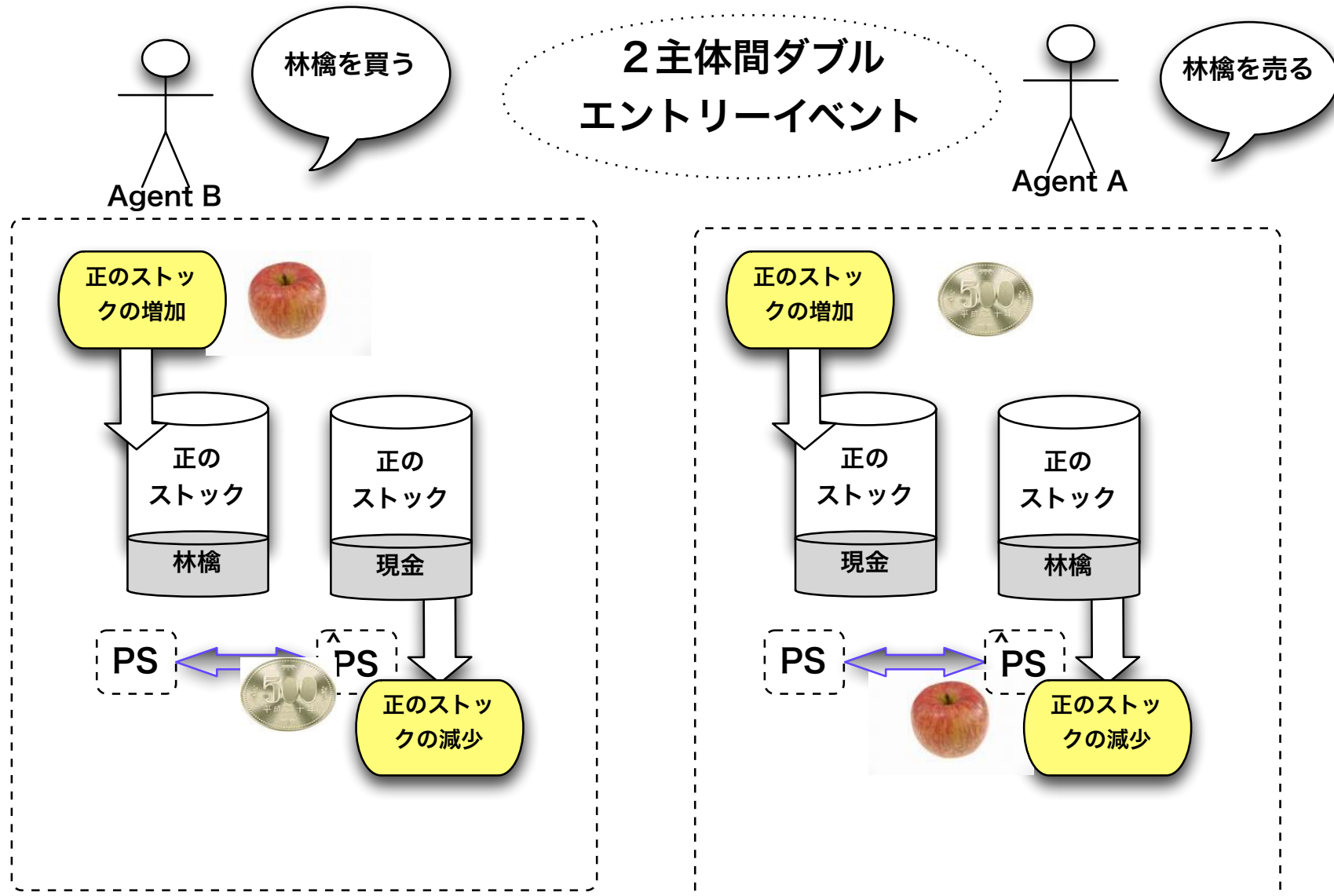
2) 複式記述では、借り方と貸し方という二つのカテゴリーの各々のイベントが1つずつ、対で発生する。

Debit Side : 借り方={Plus Stock, Out flow (損失) }

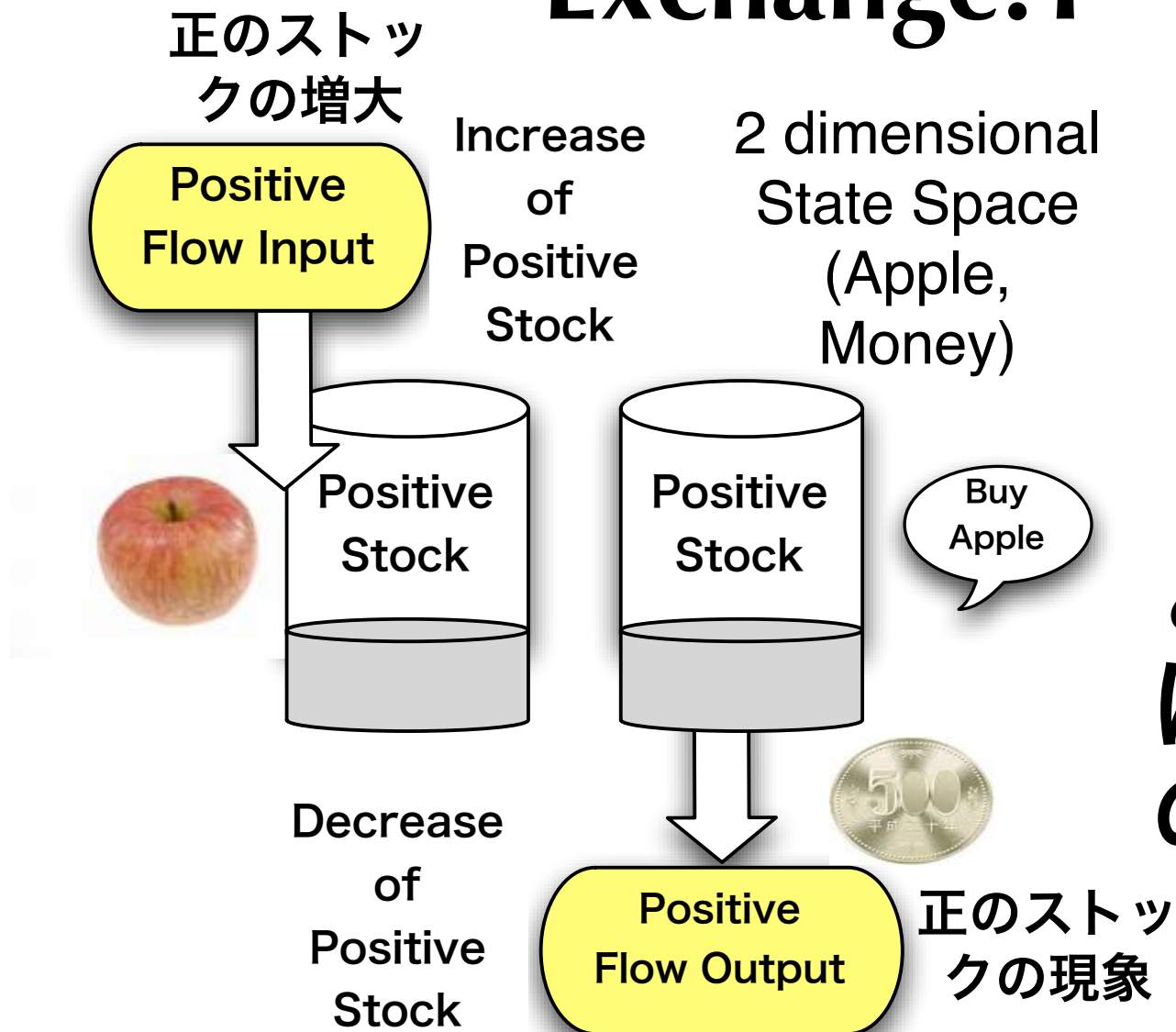
Credit Side : 貸し方={Minus Stock, Inflow (利益) , Reservers (ストック増分) }

が基本の分類。

財の購入イベント



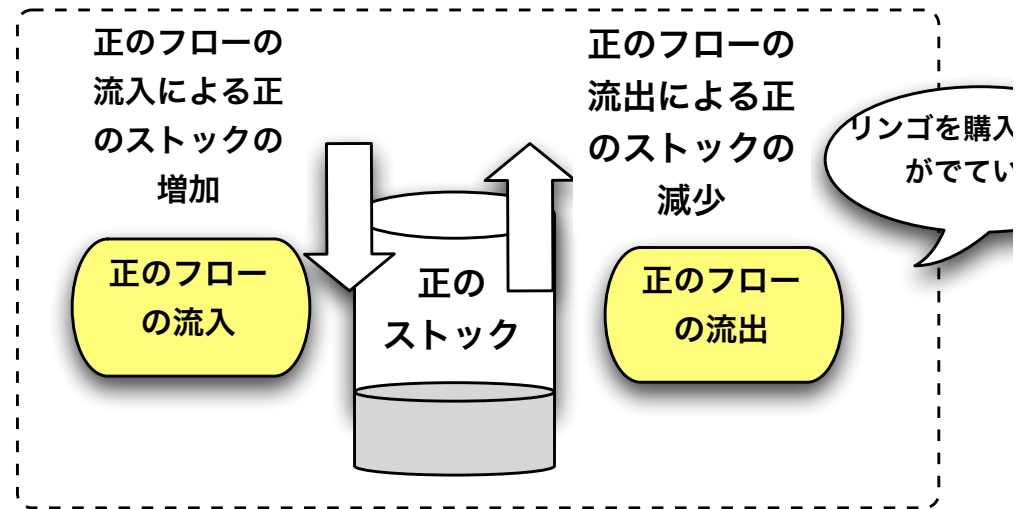
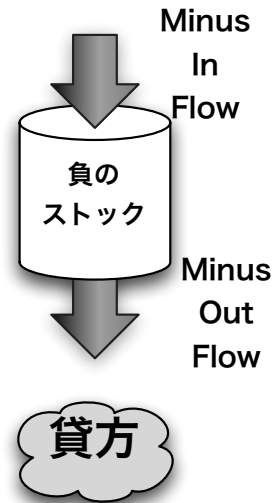
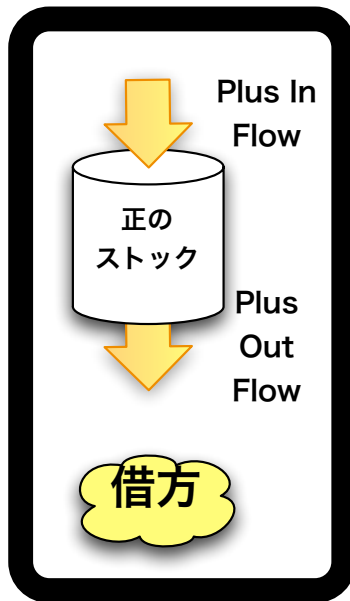
Stock and Flow Dynamics by Exchange:1



財の交換に伴う、ストックとフローのダイナミクス：1

このケースは二つの正のストックの変動

- $x = 50 \wedge \langle \text{cash} \rangle + 50 \langle \text{apple} \rangle$ リンゴを現金で買う交換



Debit Side
借方

Credit Side
貸方

50りんご

50現金

50Apple

50Cash

50^<Cash,Yen>+
50<Apple, Yen>

借金イベント

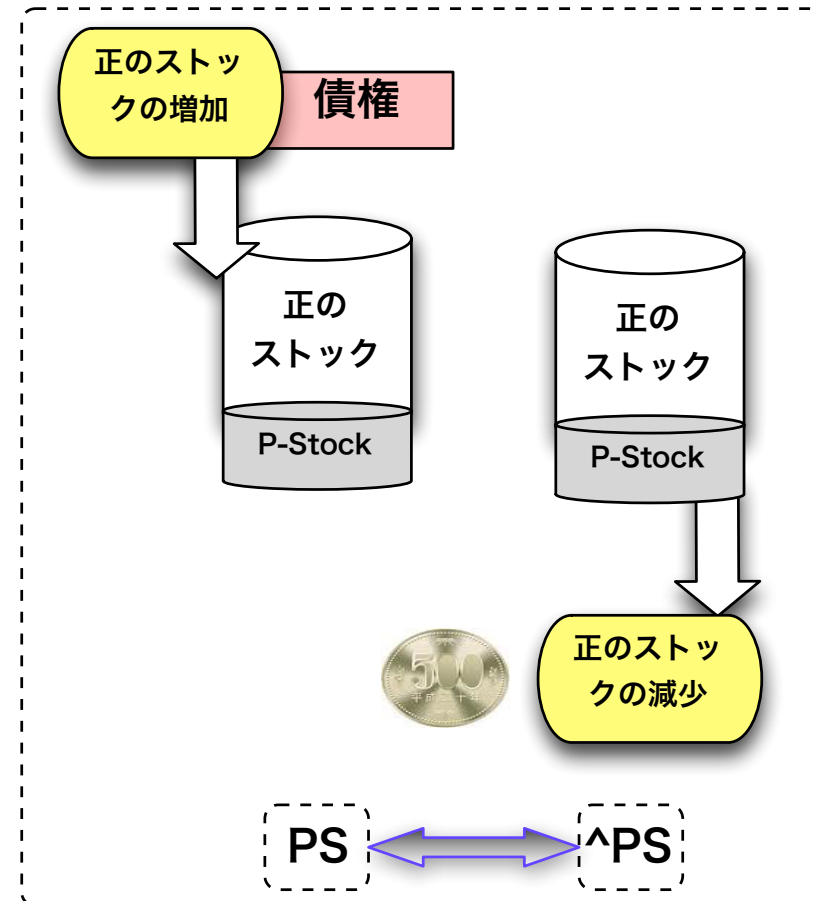
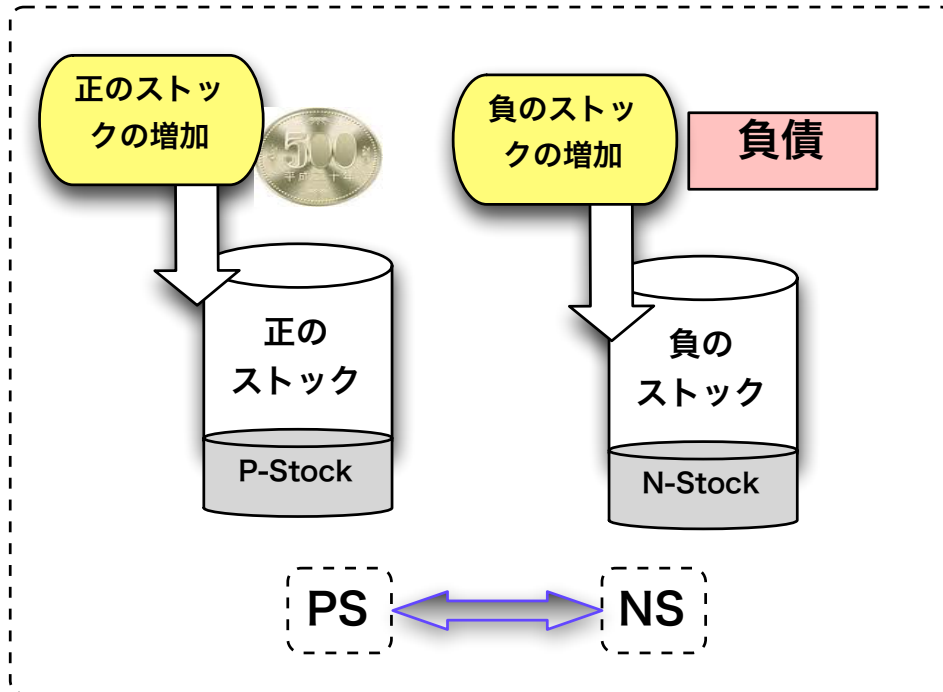
Agent B

お金を借りる

2主体間ダブル
エントリーイベント

Agent A

お金を貸す



Stock and Flow Dynamics by Exchange:3

財の交換に

伴う、ス

トックとフ

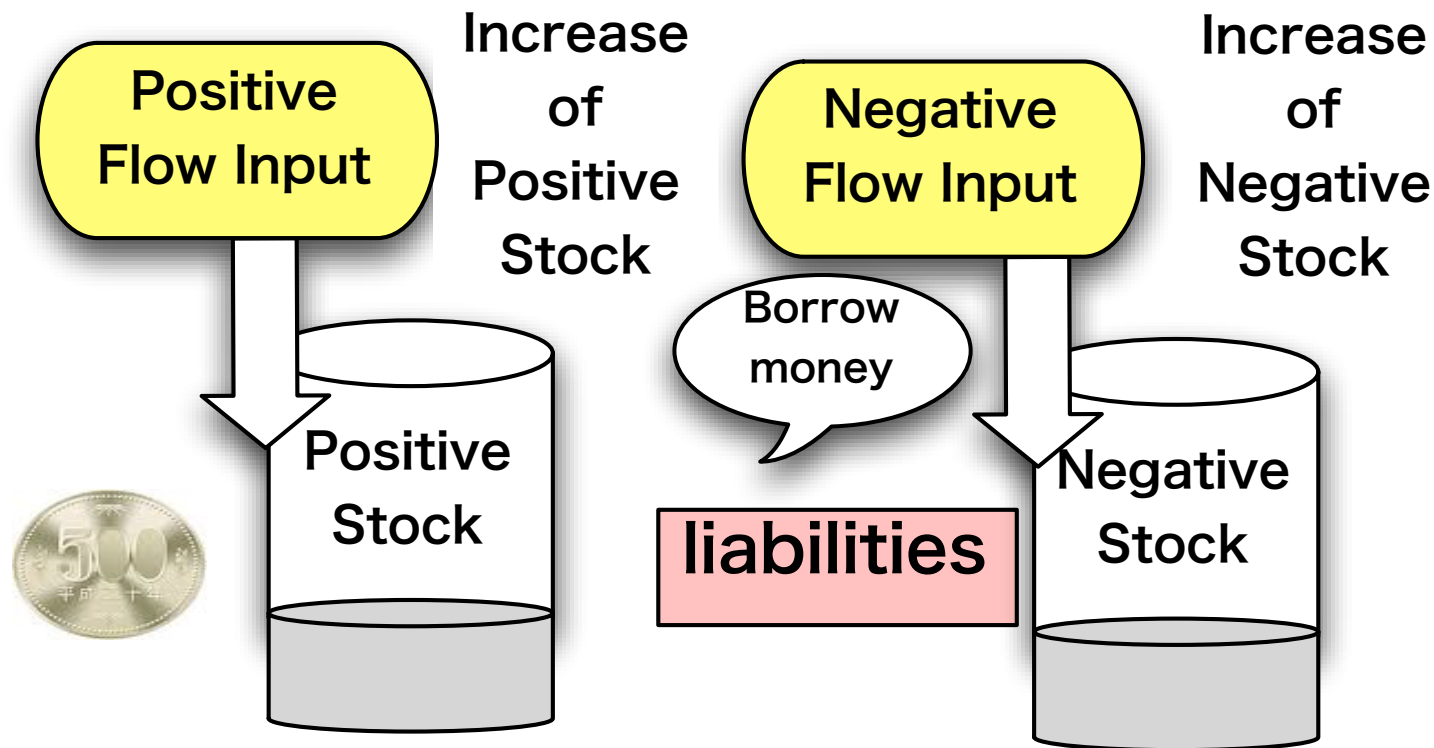
ローのダイナ

ミクス：3

負のストック
と正のストックの間の交換

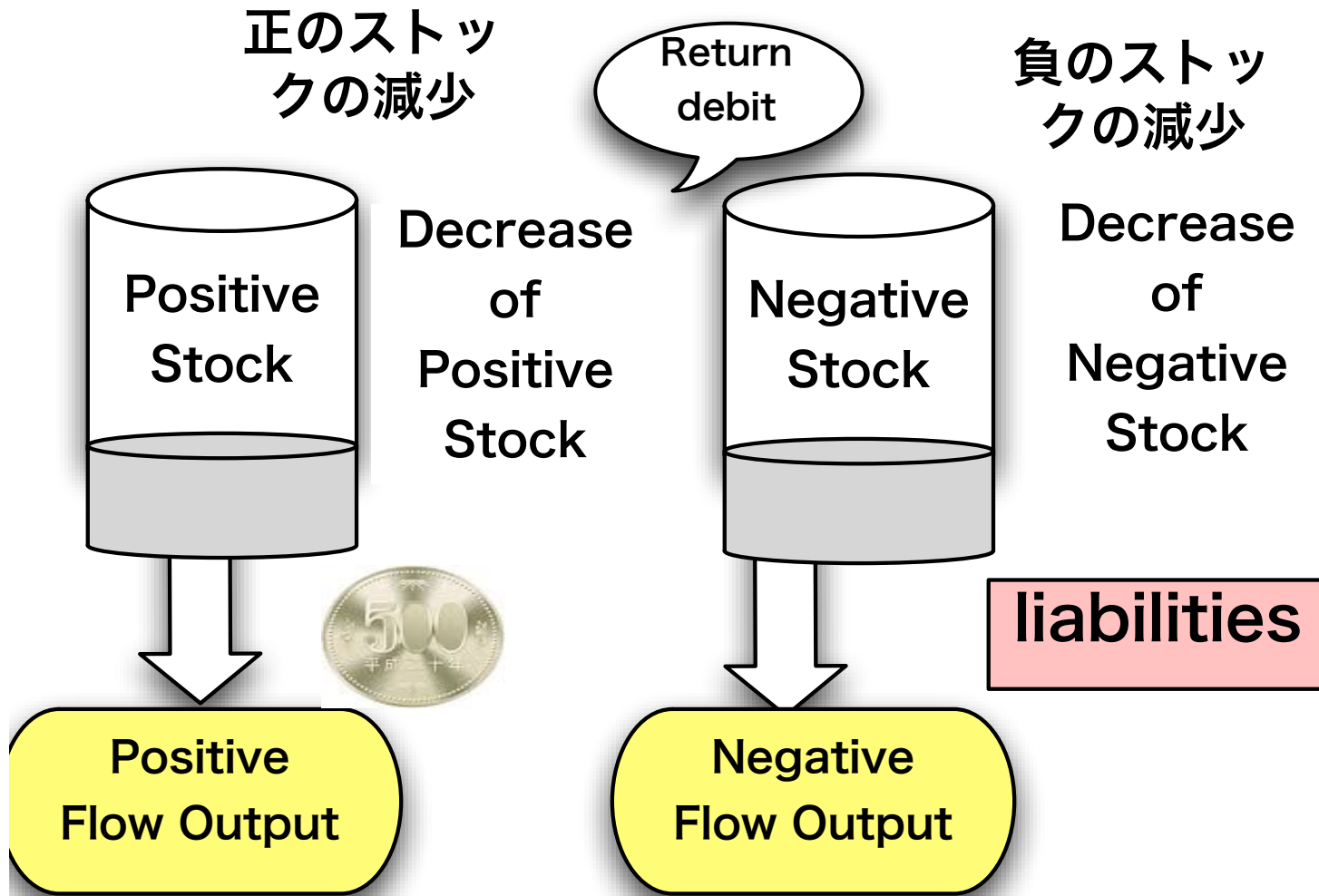
正のストックの増大

負のストックの増大



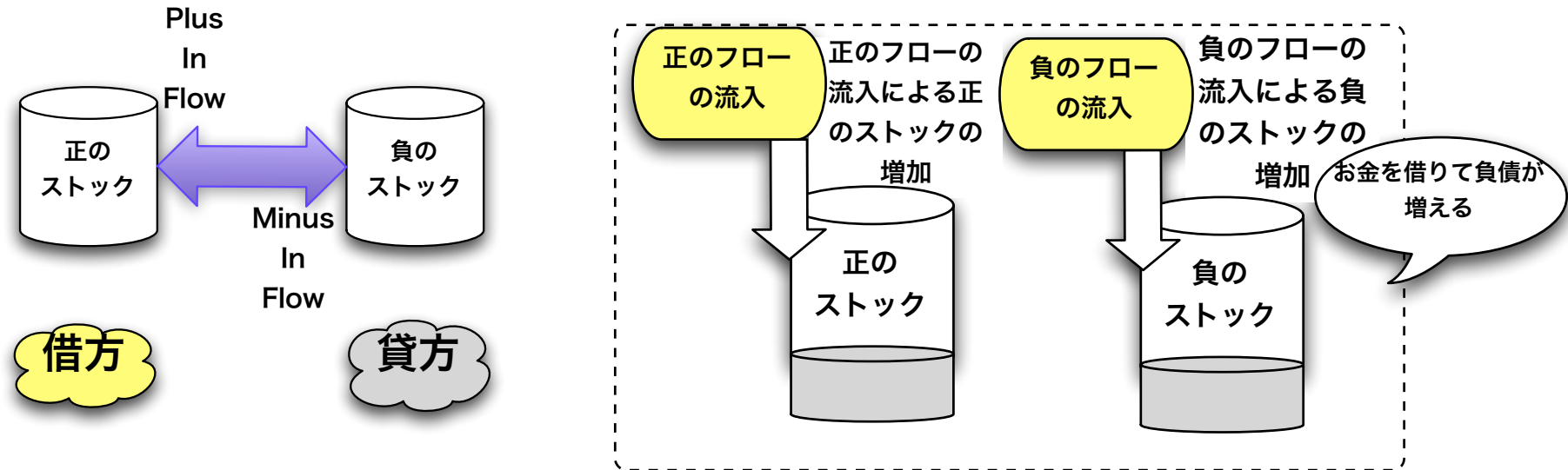
- $x=50<\text{cash}>+50<\text{liability}>$: 20円借金する

Stock and Flow Dynamics by Exchange:4



財の交換に伴う、ストックとフローのダイナミクス：4
負のストックと正のストックの間の交換

- $x = 20^{\text{cash}} + 20^{\text{liability}}$: 20円借金の返却



Debit Side
借方

Credit Side
貸方

50現金

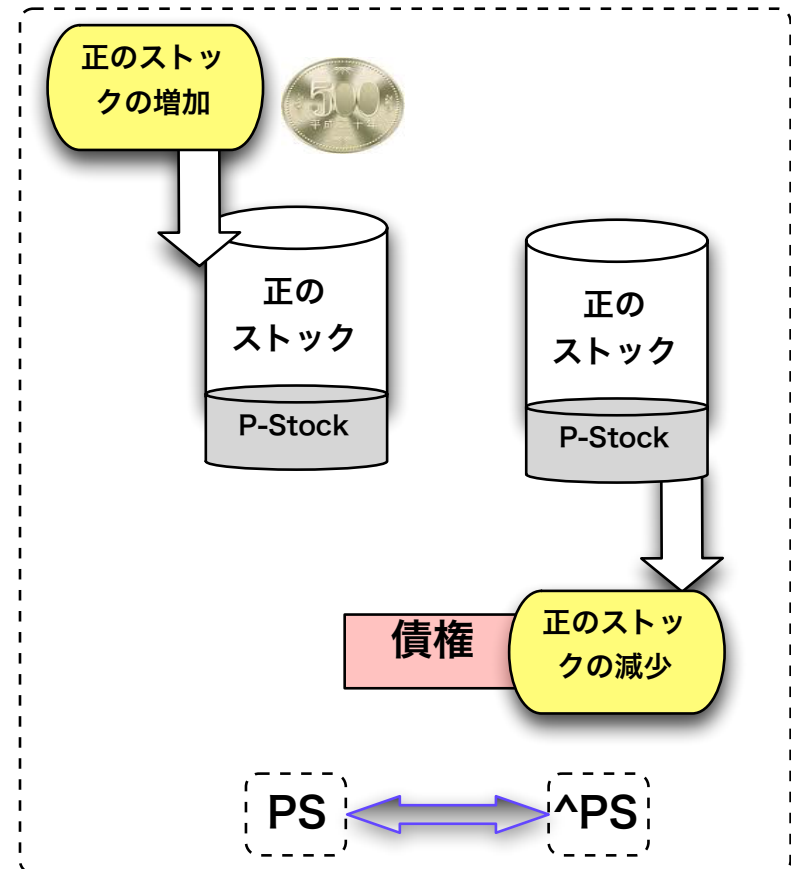
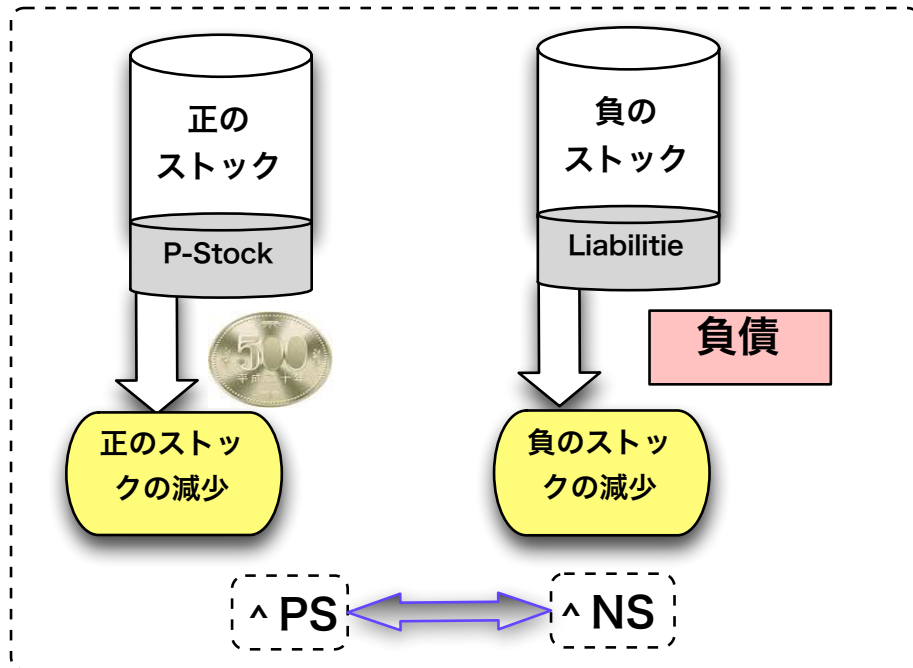
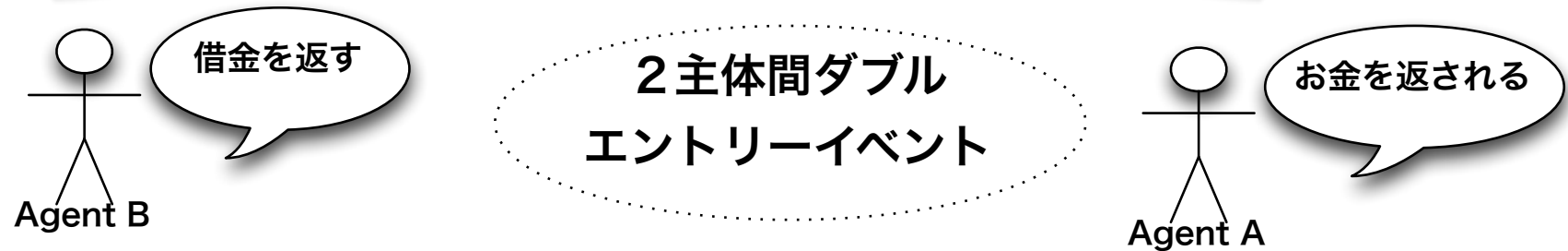
50負債

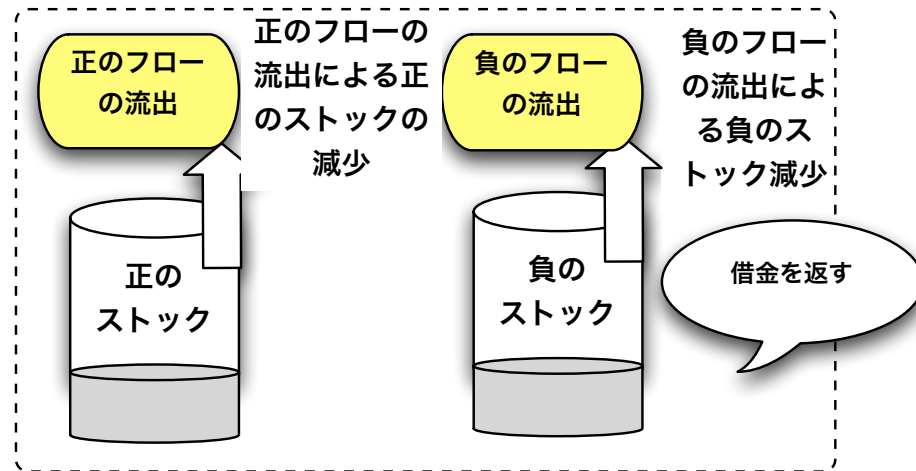
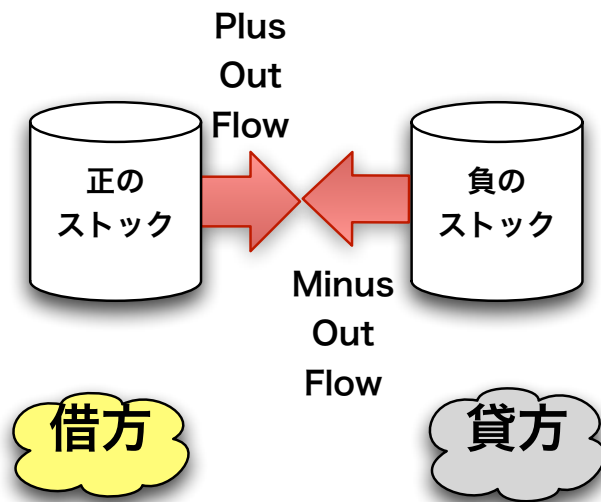
50Cash

50Liabilities

50<Cash, Yen>+
50<Liabilities, Yen>

借金返済イベント





Debit Side
借方

Credit Side
貸方

50負債

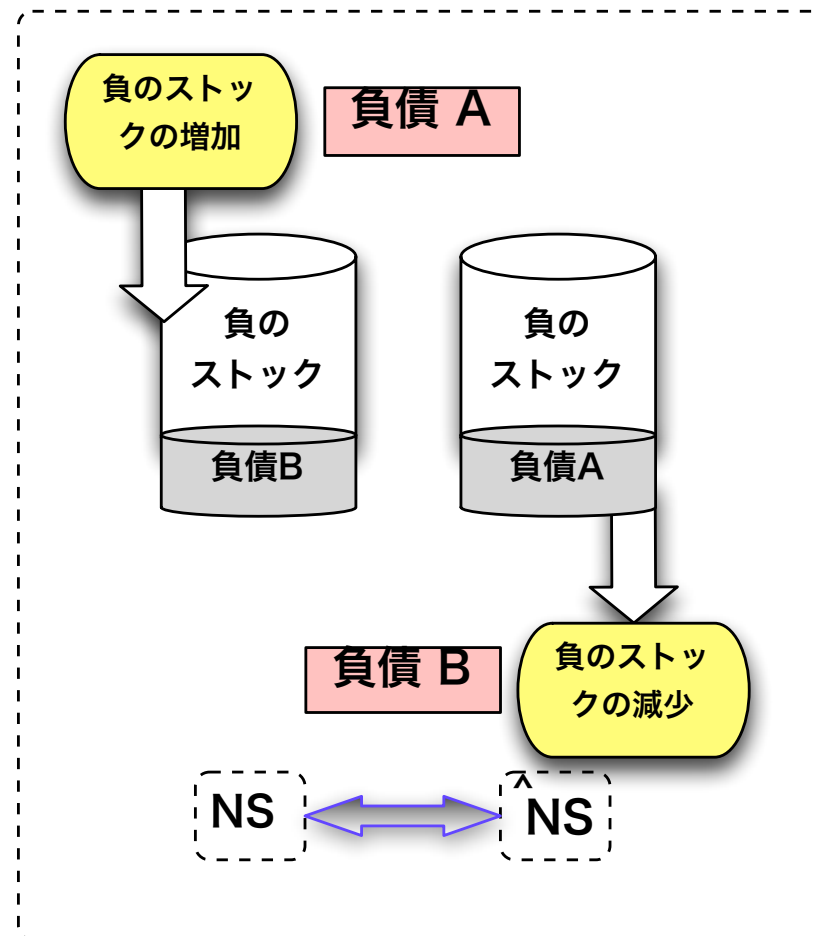
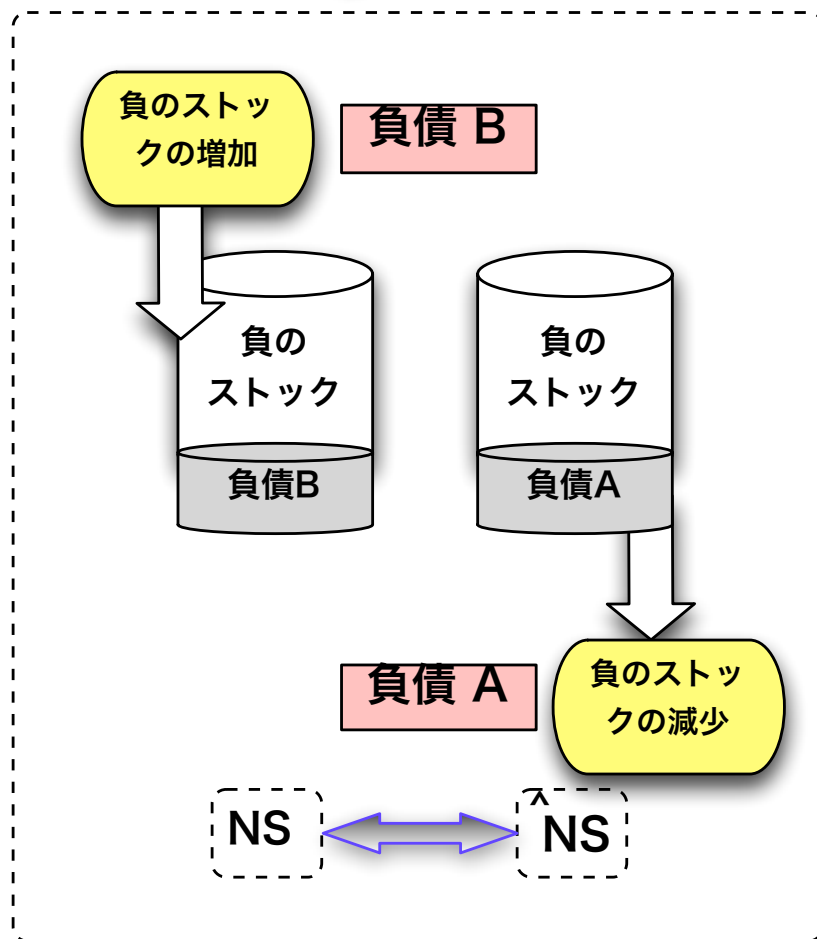
50Liabilities

50現金

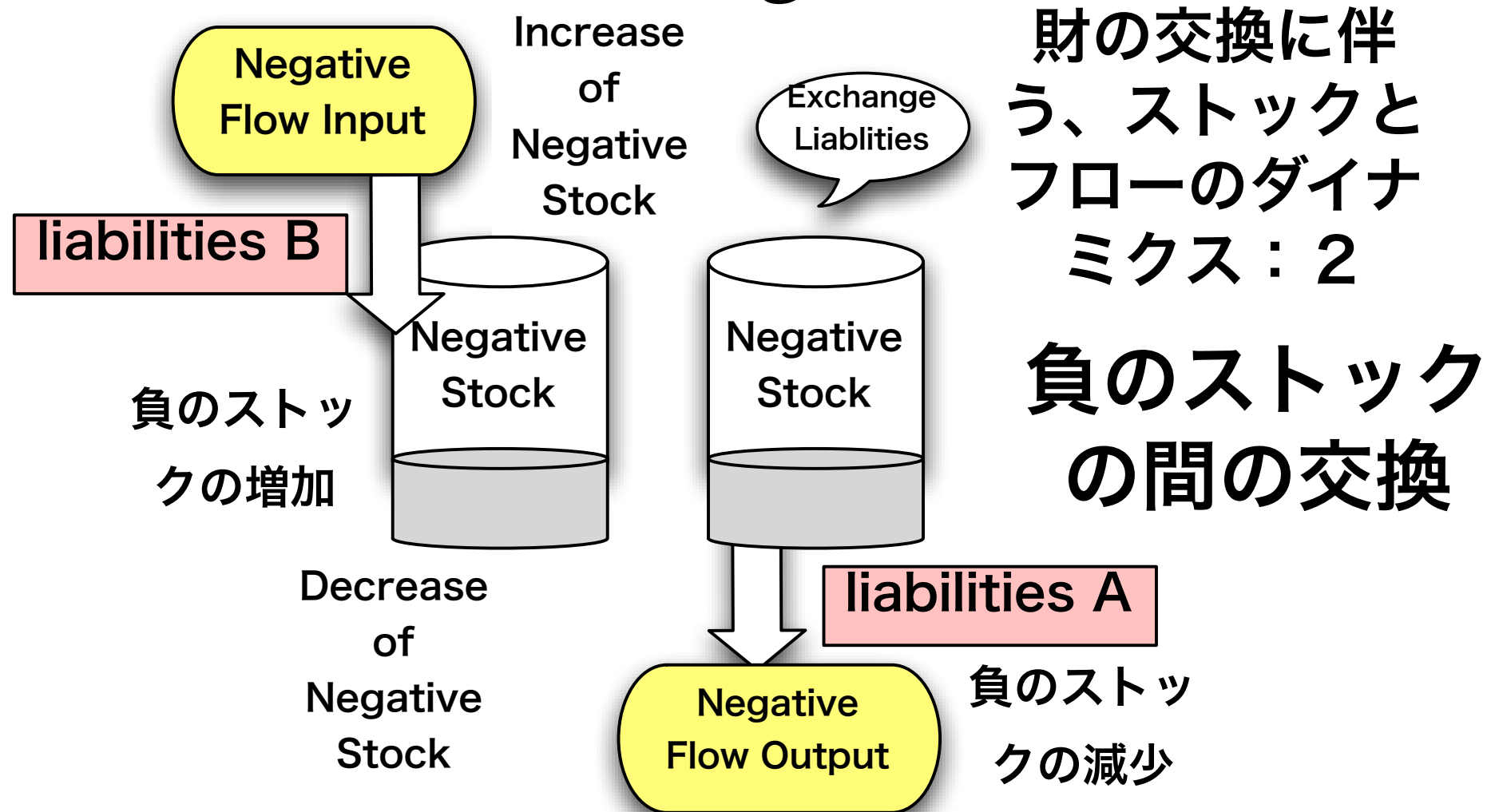
50Cash

$50^{<\text{Cash, Yen}>+}$
 $50^{<\text{Liabilities, Yen}>}$

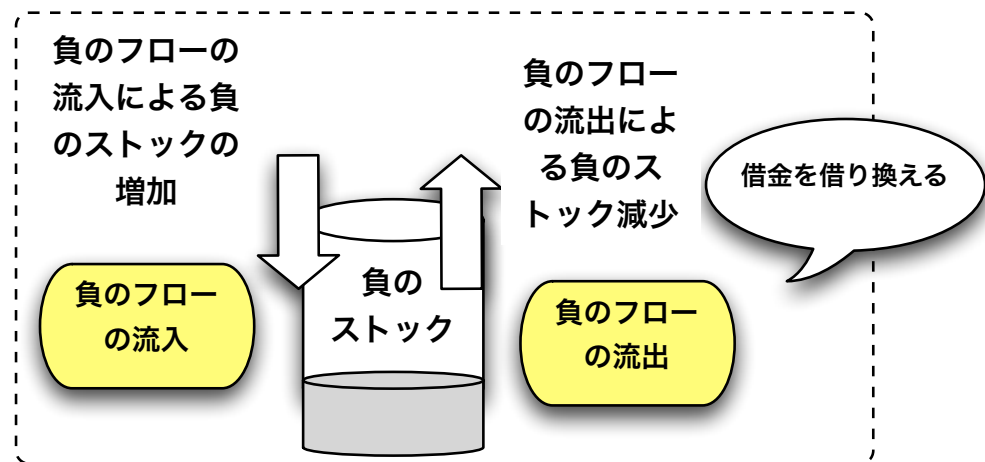
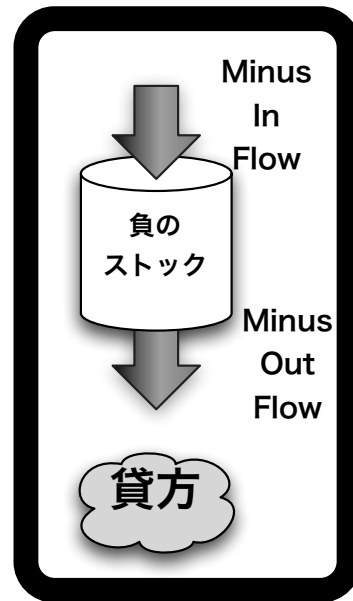
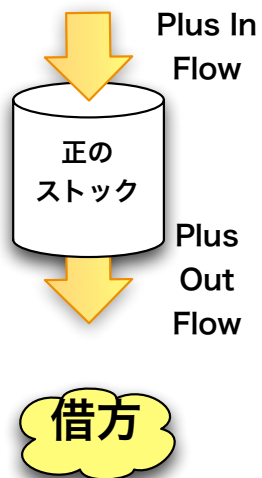
負債借り換えイベント



Stock and Flow Dynamics by Exchange:2



- $x = 20^{\wedge} \text{liability A} + 20^{\wedge} \text{liability B}$: 負債の交換



Debit Side
借方

Credit Side
貸方

50借金 1

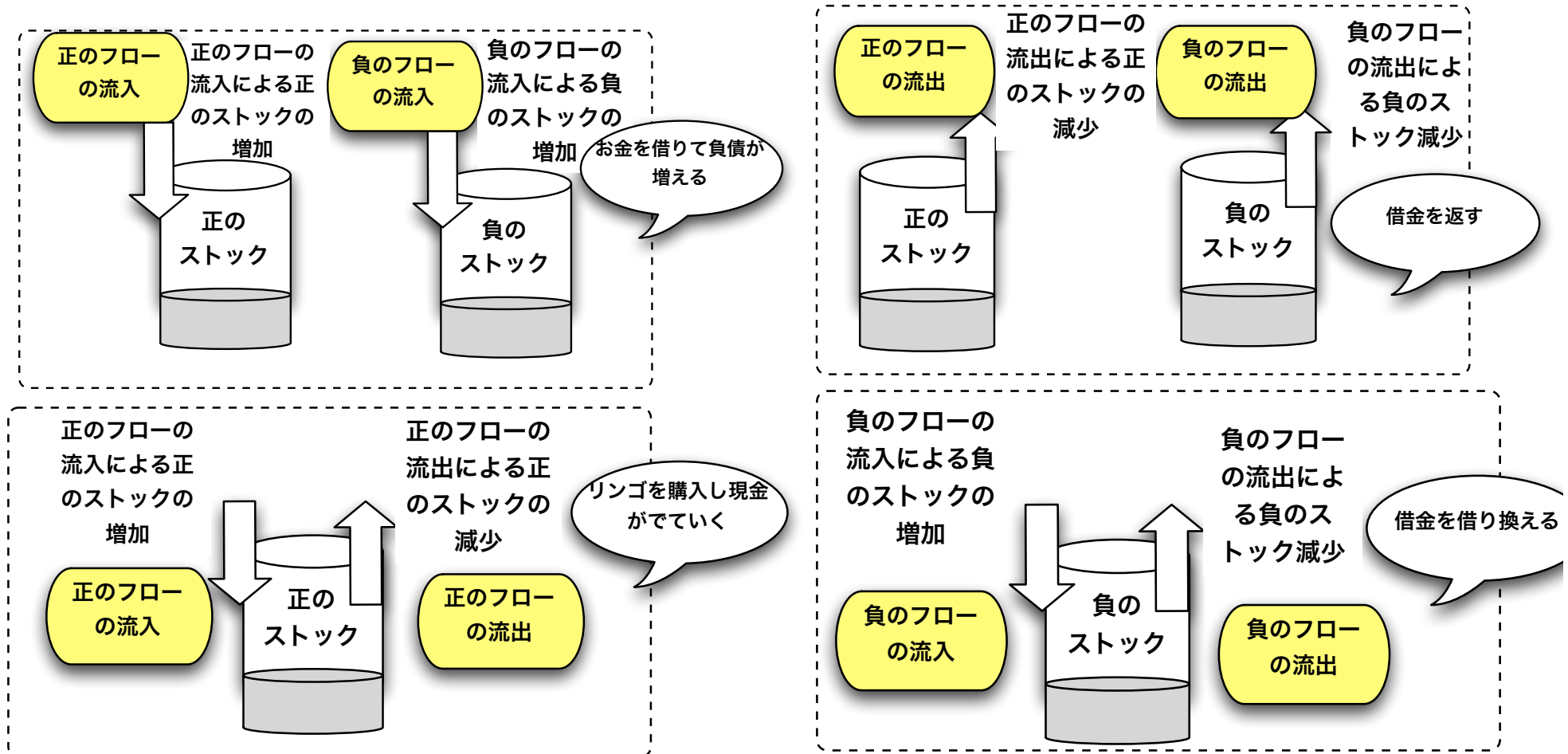
50借金 2

50Liability1

50Liability2

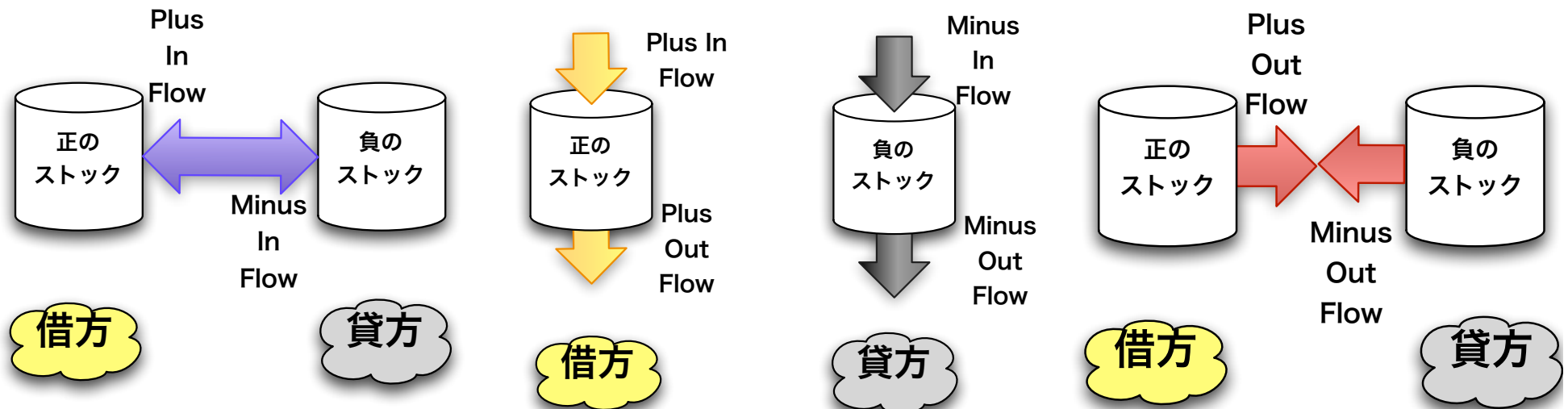
$50^{<\text{Liability1, Yen}>} + 50^{<\text{Liability2, Yen}>}$

- * 経済のシステムでは一方的に正や負のストックが増えるのではなく現金が増えて負債が増えるように取引という形で状態の変化が生じるパターンが多い（簿記的記述）。
- * 非常に多種類に分類された財や負債の状態とその変化を記述する必要が、企業や社会の財務的な状態記述とその変化の記述では必要とされる。
- * そこではさらに分類された状態の間のアグリゲーションや按分変換などの操作が求められる。これらのシステム記述を代数系として公理的に定式化した物が交換代数。
- * これを情報システムとしてプログラム言語化したものがAADL←CABSSSからダウンロード可能。



冗長代数と交換代数

- ＊ 冗長代数は負のフローとストックを扱えるようにしたベクトル空間の拡張。
- ＊ 交換代数は、簿記的なフローの変化に対する経済交換の制約条件を課した代数系。



交換代数(冗長代数)の表現の便利な所

- いろいろな分類でデータを表現できる
- プラスの数値しか出てこない。マイナスの数値の代わりに \wedge (ハット) という記号をつけた記号を使う。例： $x_1=30<\text{りんご},\#,\#,\#>$ に対して、 $x_2=20\wedge<\text{りんご},\#,\#,\#>$ は、リンゴが20減る事を意味する。
- コードではなく、解釈可能な文字で分類（基底）を表現できる。
- それに対して後述するように、振替という演算で統一的にデータ処理（計算処理）ができる。
 - それには、 $+$ 、ハット（ \wedge ）、バー（ \sim ）、射影（Projection）、ノルム（絶対値）という演算を知る必要がある。
- 例： $x_3=10<\text{みかん},\#,\#,\#>$, $x_4=50<\text{いちご},\#,\#,\#>$

交換代数（冗長代数）の操作

- 何らかの会計情報や統計情報は、分類のもととなる基底（勘定項目）とその数量の形式和で示されることが多い。
- 例： $x_1 = 30\langle\text{現金}\rangle + 20\langle\text{リンゴ}\rangle + 50\langle\text{負債}\rangle$
- $x_2 = 30\langle\text{ヒラメ}\rangle + 40\langle\text{ぶり}\rangle + 70\langle\text{現金}\rangle$ 等々である。
- これらは、 $\langle\text{分類名}\rangle$ を基底としたベクトルと見なせば、自然な演算が定義される。
- $x_1 + x_2 = 30\langle\text{現金}\rangle + 20\langle\text{リンゴ}\rangle + 50\langle\text{負債}\rangle + 30\langle\text{ヒラメ}\rangle + 40\langle\text{ぶり}\rangle + 70\langle\text{現金}\rangle$
- $= 100\langle\text{現金}\rangle + 20\langle\text{リンゴ}\rangle + 50\langle\text{負債}\rangle + 30\langle\text{ヒラメ}\rangle + 40\langle\text{ぶり}\rangle$
- ここで、マイナスの係数も許せば普通のベクトル空間となる。これに対して、マイナスの数の代わりに係数はプラスの数に限定し、基底に \wedge （ハット）という記号を導入した代数系を考える。 \wedge はオペレータの意味でも用いる（単項オペレータ）。

^操作の意味

- $y1 = x1 = 30^{<\text{現金}>} + 20^{<\text{リンゴ}>} + 50^{<\text{負債}>}$
- $y2 = x2 = 30^{<\text{ヒラメ}>} + 40^{<\text{ぶり}>} + 70^{<\text{現金}>}$
- ^は意味的には、ある項目に対して、相殺すべき反対項目を表す基底となる。
- 現金が減る事を意味するのが、 $^{<\text{現金}>}$ という基底となる。
- これを用いるとマイナスの数の代わりの表現が与えられる。ただしこの表現の方が、冗長度が高い。
- $z1 = 50^{<\text{現金}>}$
- $z2 = 60^{<\text{現金}>} + 10^{<\text{現金}>}$
- $z3 = 80^{<\text{現金}>} + 30^{<\text{現金}>}$
- などは「ある意味」同じ状態を表していると考えたい。

~作用素

- そこで相殺という操作を表す作用素（オペレータ）として、 \sim （バー）という作用素を導入する。
- $z1 = \sim z1 = \sim z2 = \sim z3 = 50<\text{現金}>$
- となる。
- $y1 + x1 = \wedge x1 + x1 = (30\wedge<\text{現金}> + 20\wedge<\text{リンゴ}> + 50\wedge<\text{負債}>) + (30<\text{現金}> + 20<\text{リンゴ}> + 50<\text{負債}>)$
- $\sim(y1 + x1) = 0$ となる。
- このようなハット(\wedge)とバー(\sim)という2つの作用を持つように拡張されたベクトル的な計算体系を冗長代数と呼ぶ。その公理的定式化は別項に述べたが、実際の計算はここで示した例が分かれば問題なく計算できる。

Λ : 分類の基底の集合

- ここで我々は、 Λ という分類の基底の集合
- 例えば、 $\Lambda = \{ \langle \text{現金} \rangle, \langle \text{リンゴ} \rangle, \langle \text{負債} \rangle, \langle \text{ヒラメ} \rangle, \langle \text{ぶり} \rangle \}$
- に対して、その上の上記のような和からなる元で構成され、バーとハットオペレーションの導入された空間を $[\Lambda]$ で示す。
- $^{\wedge}\Lambda = \{ ^{\wedge}\langle \text{現金} \rangle, ^{\wedge}\langle \text{リンゴ} \rangle, ^{\wedge}\langle \text{負債} \rangle, ^{\wedge}\langle \text{ヒラメ} \rangle, ^{\wedge}\langle \text{ぶり} \rangle \}$
- $\Lambda_{\text{ex}} = \Lambda \cup ^{\wedge}\Lambda = \{ \langle \text{現金} \rangle, \langle \text{リンゴ} \rangle, \langle \text{負債} \rangle, \langle \text{ヒラメ} \rangle, \langle \text{ぶり} \rangle, ^{\wedge}\langle \text{現金} \rangle, ^{\wedge}\langle \text{リンゴ} \rangle, ^{\wedge}\langle \text{負債} \rangle, ^{\wedge}\langle \text{ヒラメ} \rangle, ^{\wedge}\langle \text{ぶり} \rangle \}$
- $[\Lambda_{\text{ex}}] = [\Lambda \cup ^{\wedge}\Lambda]$ を Λ から生成される冗長代数（交換代数）と呼ぶ。
- 特に混乱のない限り、 $[\Lambda \cup ^{\wedge}\Lambda]$ の代わりに、 $[\Lambda]$ で冗長代数を表すこともある。

冗長代数による振替記述

- 簿記の中に振替という操作がある。これは一種の分類替えの操作である。

• 借方	金額		貸方	金額
• 現金	200		リンゴ	100
•			利益	100
• 光熱費	50		現金	50

- という八百屋さんの売上傳票を、

• 利益	100		営業収益	100	利益を営業収益＋へ
• 営業収益	50		光熱費	50	コストを営業収益－へ
• と振り替えることを考えます。					

冗長代数での振替記述

- ここでは、勘定科目から構成される基底は、価格表示、物量表示等の多様な表示を考える為に、 $\langle \text{勘定科目}, \text{種別} \rangle$ とする。上記の取引は円表示で、
- $x_1 = 200 \langle \text{現金}, \text{円} \rangle + 100^\wedge \langle \text{リンゴ}, \text{円} \rangle + 100 \langle \text{利益}, \text{円} \rangle$
- $x_2 = 50 \langle \text{光熱費}, \text{円} \rangle + 50^\wedge \langle \text{現金}, \text{円} \rangle$
- $x_3 = 100^\wedge \langle \text{利益}, \text{円} \rangle + 100 \langle \text{営業収益}, \text{円} \rangle$: 振替データ
- $x_4 = 50^\wedge \langle \text{営業収益}, \text{円} \rangle + 50^\wedge \langle \text{光熱費}, \text{円} \rangle$: 振替データ
- $y = x_1 + x_2 + x_3 + x_4 = 200 \langle \text{現金}, \text{円} \rangle + 100^\wedge \langle \text{リンゴ}, \text{円} \rangle + 100 \langle \text{利益}, \text{円} \rangle + 50 \langle \text{光熱費}, \text{円} \rangle + 50^\wedge \langle \text{現金}, \text{円} \rangle + 100^\wedge \langle \text{利益}, \text{円} \rangle + 100 \langle \text{営業収益}, \text{円} \rangle + 50^\wedge \langle \text{営業収益}, \text{円} \rangle + 50^\wedge \langle \text{光熱費}, \text{円} \rangle$
- $\sim y = 150 \langle \text{現金}, \text{円} \rangle + 100^\wedge \langle \text{リンゴ}, \text{円} \rangle + 50 \langle \text{営業収益}, \text{円} \rangle$: 残高試算表
- \sim (バー) は相殺計算を行う交換代数 (冗長代数) 上のオペレータである。

簿記による表現

- 借方 | 貸方
- =====
- cash 2 0 0 | apple 1 0 0
- | profit 1 0 0
- cost 5 0 | cash 5 0
- -----

- 振替取引
- profit 1 0 0 | revenue 1 0 0
- revenue 5 0 | cost 5 0
- -----

- 残高試算表
- cash 1 5 0 | apple 1 0 0
- | revenue 5 0

振替という一般演算

- 振替という概念で極めて一般的なデータに対する演算が構成できる。
- 振替操作は
 - \sim (振替対象の元データ + 振替データ)で示される
- (1) アグリゲーション
- (2) 按分 (分割)
- (3) 単位の変換、などいろいろ利用できる
 - e.x. change between “yen” and “dollar”

振替という操作 2

- この振替という操作を用いる事で、冗長代数上の様々な変換作業が比較的統一的、かつ容易に行える。以下次の4つの変換に対して、これを振替と見なす作業を示す。
- (1) アグリゲーション (合併)
- (2) 按分
- (3) 単位の変換
- (4) 通貨の変換
- (5) インフレなどの価格体系の変換

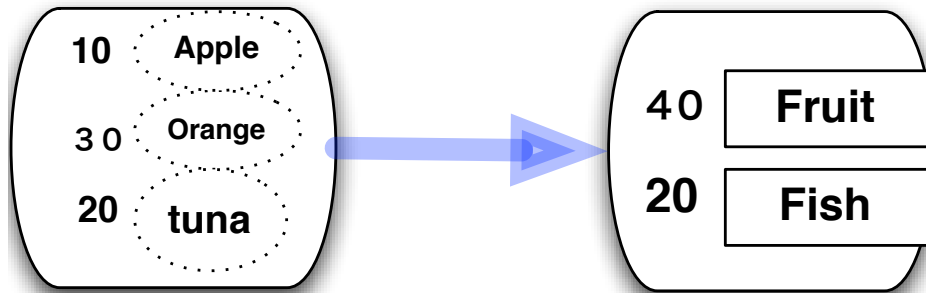
(1) アグリゲーション

- 300円の津軽というリンゴの品種と、200円の富士、100円の紅玉があったとする。これを
- $x=300<\text{津軽,円}>+200<\text{富士,円}>+100<\text{紅玉,円}>$ と表現する。
- この津軽、富士、紅玉という品種をリンゴとまとめて分類する操作も一種の振替となる。
- $\{\text{津軽、富士、紅玉}\} \rightarrow \{\text{リンゴ}\}$ という対応関係のマップが与えられているとする。
- $F(x)=300^{\wedge}<\text{津軽,円}>+200^{\wedge}<\text{富士,円}>+100^{\wedge}<\text{紅玉,円}>+300<\text{リンゴ,円}>+200<\text{リンゴ,円}>+100<\text{リンゴ,円}>$ という元を生成する。
- アグリゲーションを表す振替 $G(x)$ そのものは、 $F(x)$ を用いて、 $\sim\{x+F(x)\}$ で与えられる。
- $\sim\{x+F(x)\}=\sim\{300<\text{津軽,円}>+200<\text{富士,円}>+100<\text{紅玉,円}>+300^{\wedge}<\text{津軽,円}>+200^{\wedge}<\text{富士,円}>+100^{\wedge}<\text{紅玉,円}>+300<\text{リンゴ,円}>+200<\text{リンゴ,円}>+100<\text{リンゴ,円}>\}=600<\text{リンゴ,円}>$ となる。

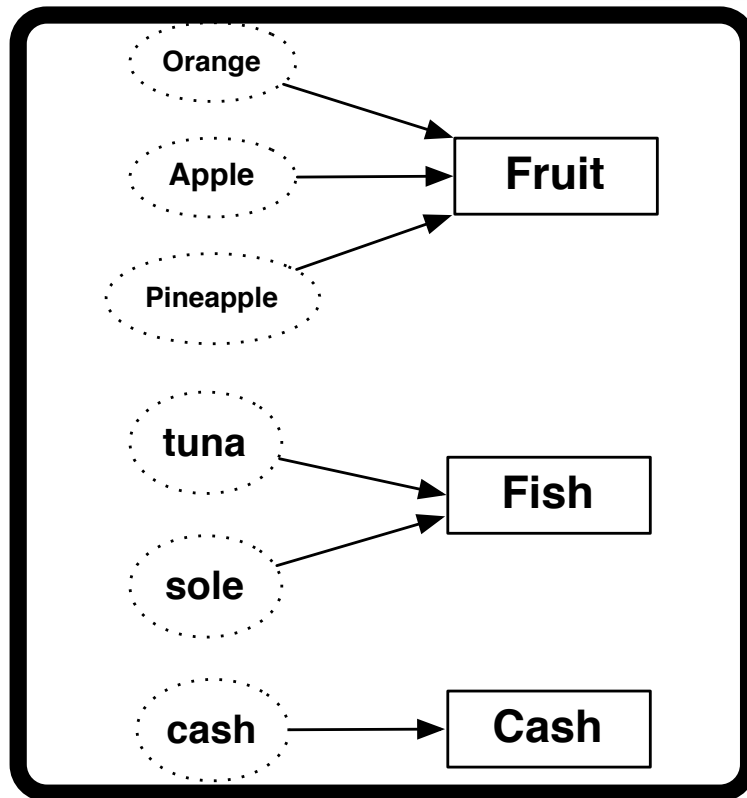
アグリゲーション続き

- これは一般に $\Lambda \rightarrow \Lambda'$ に対して、変換 f が onto (全射) で与えられているとき、
- $x = \Sigma \{ a(e) \langle e, \text{円} \rangle \mid e \in \Lambda \}$ に対して、
- $F[f][g](x) = \Sigma \{ a(e) \langle e, \text{円} \rangle + a(e) \langle g, \text{円} \rangle \mid e \in f^{-1}(g) \}$
- ただし、 $g \in \Lambda'$
- $F[f](x) = \Sigma \{ F[f][g](x) \mid g \in \Lambda' \}$ ただし、 $x \in [\Lambda]$
- アグリゲーションマップは、
- $G[f](x) = \sim \{ x + F[f](x) \}$ ただし、 $x \in [\Lambda]$
- で定式化される。

(1) Aggregation



- $x = 10\langle\text{apple}\rangle + 30\langle\text{orange}\rangle + 20\langle\text{tuna}\rangle$:Original data

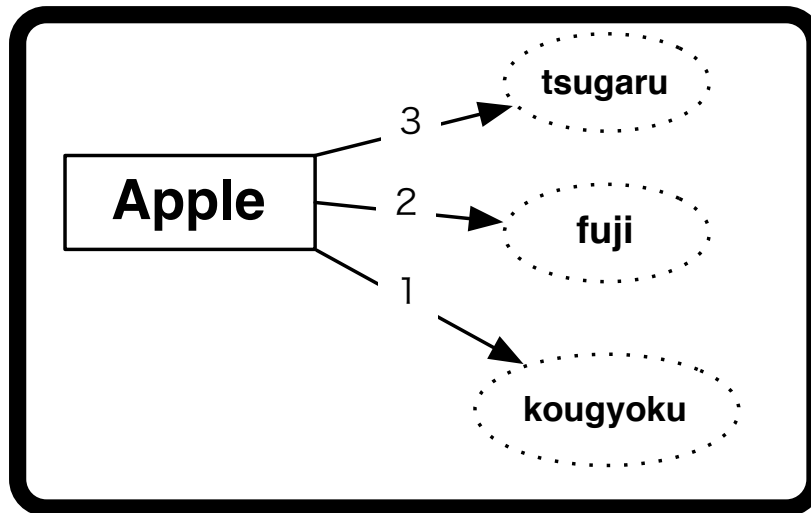
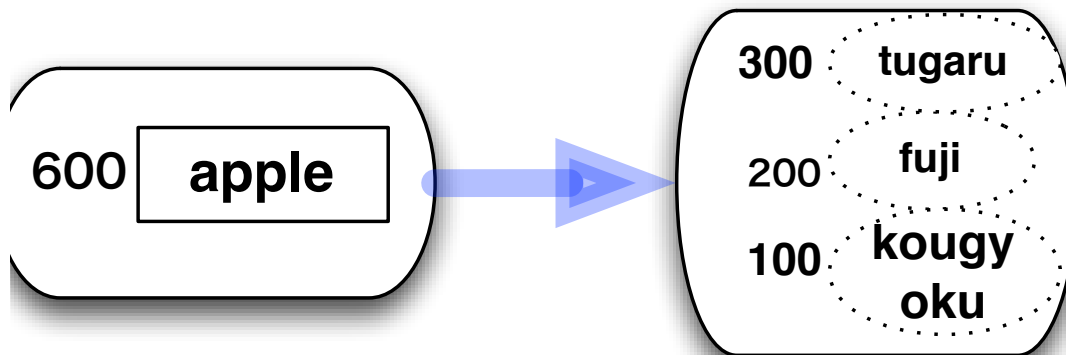


- $y = 10^{\langle\text{apple}\rangle} + 10\langle\text{Fruit}\rangle + 30^{\langle\text{orange}\rangle} + 30\langle\text{Fruit}\rangle + 20\langle\text{tuna}\rangle + 20\langle\text{fish}\rangle$:振替データ：リンゴとオレンジをフルーツに、ツナを魚に振替る。
- $z = \sim(x+y) = 40\langle\text{Fruit}\rangle + 20\langle\text{fish}\rangle$
- 振替データはアグリゲーションの関係から導かれる。

(2) 按分

- 按分とは、一つの分類項目を更に細かく複数の分類項目へと分割することである。例えば、上記の例だと、リンゴを更に細かい品種である、津軽、富士、紅玉に分割することを意味する。
- 仮に、 $600<\text{リンゴ}, \text{円}>$ が与えられたとき、それを按分しようとする、それぞれの細目への按分比率が与えられている必要がある。
- $\{\text{リンゴ}\} \rightarrow \{\text{津軽}, \text{富士}, \text{紅玉}\}$ に対して按分比率が、1:1:1であったとするなら、 $x=600<\text{リンゴ}, \text{円}>$ に対して、
- $F(x)=600^{\wedge}<\text{リンゴ}, \text{円}>+200<\text{津軽}, \text{円}>+200<\text{富士}, \text{円}>+200<\text{紅玉}, \text{円}>$
- 按分を表す振替 $G(x)$ は
- $G(x)=\sim\{x+F(x)\}=\sim\{600<\text{リンゴ}, \text{円}>+600^{\wedge}<\text{リンゴ}, \text{円}>+200<\text{津軽}, \text{円}>+200<\text{富士}, \text{円}>+200<\text{紅玉}, \text{円}>\}=200<\text{津軽}, \text{円}>+200<\text{富士}, \text{円}>+200<\text{紅玉}, \text{円}>$ となる。

(2) 按分 (分割)



- $x=600\langle\text{apple},\text{yen}\rangle$
:Original Data
- $y=600^{\wedge}\langle\text{apple},\text{yen}\rangle+300\langle\text{tugaru},\text{yen}\rangle+200\langle\text{fuji},\text{yen}\rangle+100\langle\text{kougyoku},\text{yen}\rangle$:振替データ、リンゴを3:2:1で津軽と富士と紅玉に分割する。
- $z=\sim(x+y)=300\langle\text{tugaru},\text{yen}\rangle+200\langle\text{fuji},\text{yen}\rangle+100\langle\text{kougyoku},\text{yen}\rangle$
- 振替データは分割の関係と分割比率から構成される。

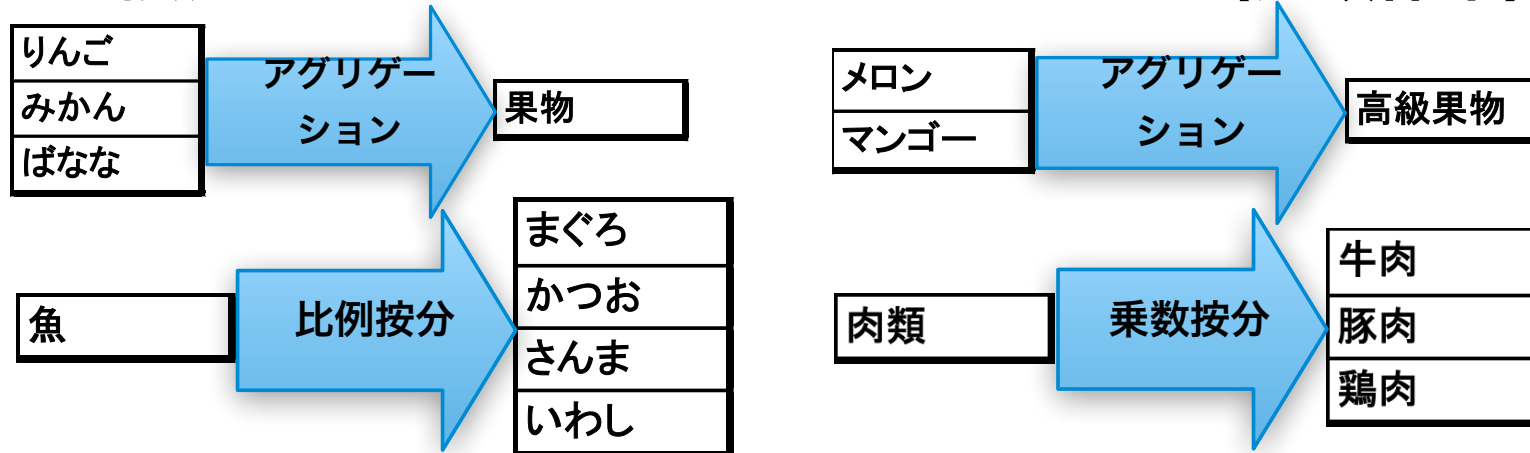
比率による按分

- 配分比率の重みを表す係数 $w(<\text{津軽}, \text{円}>)$, $w(<\text{富士}, \text{円}>)$, $w(<\text{紅玉}, \text{円}>)$ を想定する。 $w(<\text{津軽}, \text{円}>) + w(<\text{富士}, \text{円}>) + w(<\text{紅玉}, \text{円}>) = 1$ とする。
- 例えば、比率データ、 $W(<\text{津軽}, \text{円}>) = 3$ 、 $W(<\text{富士}, \text{円}>) = 2$ 、 $W(<\text{紅玉}, \text{円}>) = 1$ 与えると、 $W_{\text{sum}} = W(<\text{津軽}, \text{円}>) + W(<\text{富士}, \text{円}>) + W(<\text{紅玉}, \text{円}>) = 6$ となり、
- 係数 $w(<\text{津軽}, \text{円}>) = W(<\text{津軽}, \text{円}>) / W_{\text{sum}} = 3/6 = 1/2$
- $w(<\text{富士}, \text{円}>) = W(<\text{富士}, \text{円}>) / W_{\text{sum}} = 2/6 = 1/3$
- $w(<\text{紅玉}, \text{円}>) = W(<\text{紅玉}, \text{円}>) / W_{\text{sum}} = 1/6$
- $F(x) = F(600<\text{リンゴ}, \text{円}>) = 600^{\text{リンゴ}, \text{円}} + w(<\text{津軽}, \text{円}>) \times 600<\text{津軽}, \text{円}> + w(<\text{富士}, \text{円}>) \times 600<\text{富士}, \text{円}> + w(<\text{紅玉}, \text{円}>) \times 600<\text{紅玉}, \text{円}>$
- $= 600^{\text{リンゴ}, \text{円}} + 300<\text{津軽}, \text{円}> + 200<\text{富士}, \text{円}> + 100<\text{紅玉}, \text{円}>$
- $G(x) = \sim\{x + F(x)\} = \sim\{600<\text{リンゴ}, \text{円}> + 600^{\text{リンゴ}, \text{円}} + 300<\text{津軽}, \text{円}> + 200<\text{富士}, \text{円}> + 100<\text{紅玉}, \text{円}>\}$

按分の一般形式

- これは一般に、 $\Lambda' \rightarrow \Lambda$ に f が onto (全射) で与えられているとき、按分を Λ から Λ' の方向で考えたい。これは $e \in \Lambda$ に対して、 $g_1[e], \dots, g_m[e] \in f^{-1}(e)$, $e \in \Lambda$, $e \in \Lambda$ が、 $g_1[e], \dots, g_m[e]$ に按分されることを意味する。
- このとき、按分の相対比率データ、 $W[e] = \{W(g_1[e]), \dots, W(g_m[e])\}$ が与えられれば、按分比率データ $w[e] = \{w(g_1[e]), \dots, w(g_m[e])\}$ は次のように与えられる。
- $W_{\text{sum}}[f^{-1}(e)] = \sum \{ g \mid g \in f^{-1}(e) \}$
- $w(g) = W(g) / W_{\text{sum}}[f^{-1}(e)]$ ただし、 $g \in f^{-1}(e)$, $e \in \Lambda$
- そこで、一般に $[\Lambda]$ の元、 $x \in [\Lambda]$ に対して、按分操作は次のように定義される。
- $F[f, w][e](x) = \sum \{ w(g)g \mid g \in f^{-1}(e) \}$ ただし $e \in \Lambda$
- $F[f, w](x) = \sum \{ F[f][e](x) \mid e \in \Lambda \}$ ただし、 $x \in [\Lambda]$
- ここで按分マップは、
- $G[f, w](x) = \sim \{ x + F[f, w](x) \}$ ただし、 $x \in [\Lambda]$

按分とアグリゲーションによるコード（分類体系）の振替変換



from_name	from_unit	from_time	from_subject	to_name	to_unit	to_time	to_subject	attribute	value
# NormalExTransfer.csv - 2009/06/16									
りんご				果物	個			aggre	
みかん				果物	個			AgGrE	
ばなな				果物	個			AGGRE	
メロン				高級果物	個				
マンゴー				高級果物	個				
魚				まぐろ				ratio	5
魚				かつお				RaTiO	3
魚				さんま				RATIO	1
魚				いわし				ratio	1
肉類				牛肉				multiply	2
肉類				豚肉				MULTIPLY	1
肉類				鶏肉				MULTIPLY	1

比例按分 5:3:1:1

乗数按分 2:1:1

按分とアグリゲーションによる会計データの振替変換の事例

10	NO_HAT	りんご	円	#	#
20	NO_HAT	みかん	円	#	#
30	NO_HAT	ばなな	円	#	#
100	NO_HAT	魚	円	#	#
5	NO_HAT	肉類	円	#	食肉

10	NO_HAT	りんご	円	#	#
20	NO_HAT	みかん	円	#	#
30	NO_HAT	ばなな	円	#	#
100	NO_HAT	魚	円	#	#
5	NO_HAT	肉類	円	#	食肉
10	HAT	りんご	円	#	#
60	NO_HAT	果物	個	#	#
20	HAT	みかん	円	#	#
30	HAT	ばなな	円	#	#
100	HAT	魚	円	#	#
50	NO_HAT	まぐろ	円	#	#
30	NO_HAT	かつお	円	#	#
10	NO_HAT	さんま	円	#	#
10	NO_HAT	いわし	円	#	#
5	HAT	肉類	円	#	食肉
10	NO_HAT	牛肉	円	#	食肉
5	NO_HAT	豚肉	円	#	食肉
5	NO_HAT	鶏肉	円	#	食肉

60	NO_HAT	果物	個	#	#
50	NO_HAT	まぐろ	円	#	#
30	NO_HAT	かつお	円	#	#
10	NO_HAT	さんま	円	#	#
10	NO_HAT	いわし	円	#	#
10	NO_HAT	牛肉	円	#	食肉
5	NO_HAT	豚肉	円	#	食肉
5	NO_HAT	鶏肉	円	#	食肉

借方	単位円	貸方	単位円
りんご	10		
みかん	20		
ばなな	30		
魚	100		
肉類	5		

アグリゲーション

果物	60	りんご	10
		みかん	20
		ばなな	30

まぐろ	50	魚	100
かつお	30		
さんま	10		
いわし	10		

比例按分

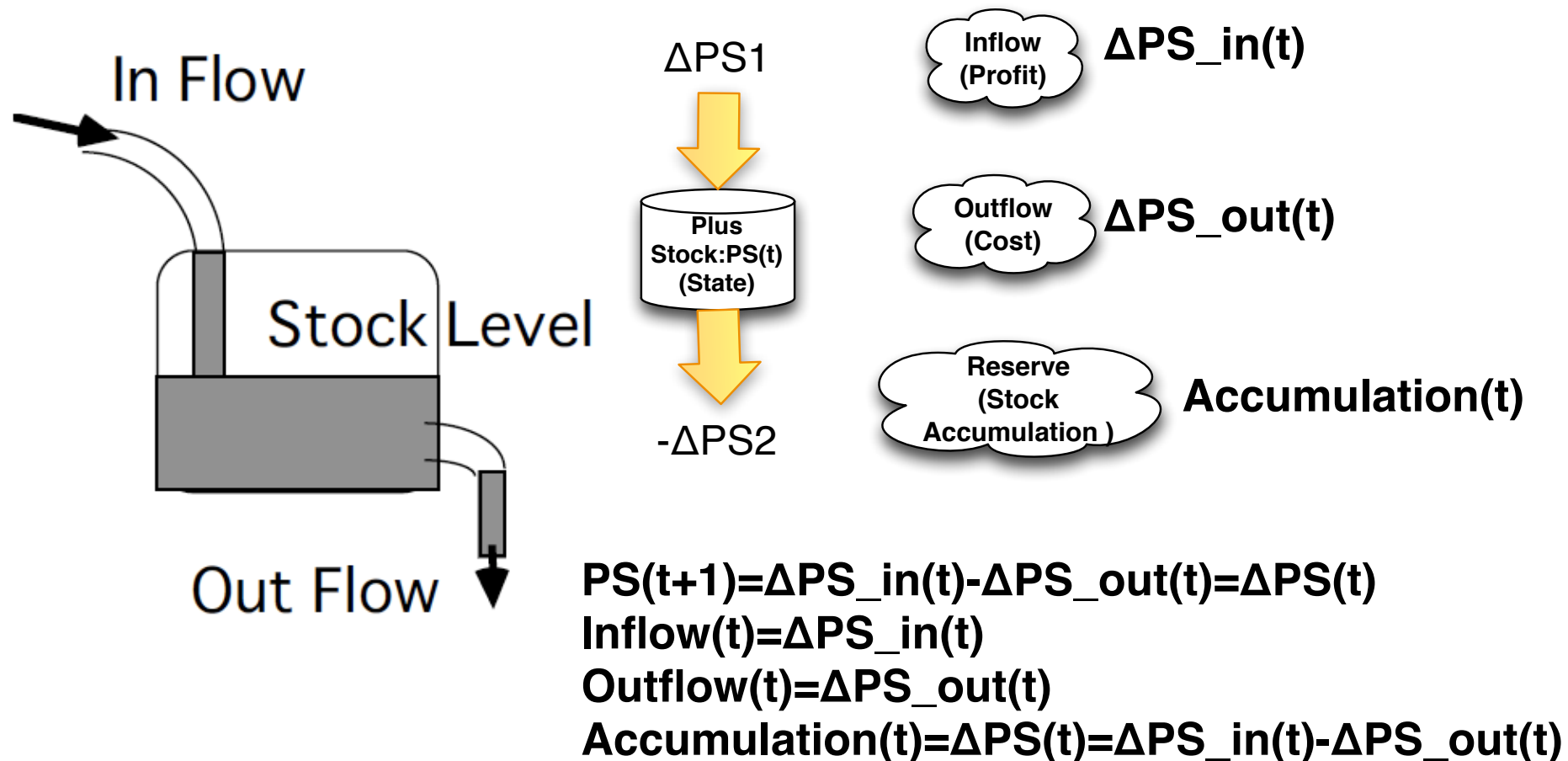
5:3:1:1

牛肉	10	肉類	5
豚肉	5		
鶏肉	5		

乗数按分 2:1:1

借方	単位円	貸方	単位円
果物	60		
まぐろ	50		
かつお	30		
さんま	10		
いわし	10		
牛肉	10		
豚肉	5		
鶏肉	5		

自然科学的なストックフロー系もダブルエン トリーで記述する事ができる



【1次元ウォーターフローの状態変数での離散時間交換代数プラス&マイナスストック型ダブルエントリー・ストック・フロー記述例】

(期首残高) PSReserve[i](t)：期首の状態のダブルエントリー記述

ここではWaterについて、初期状態をダブルエントリーで与えている。

$$PS(t)=30\langle \text{Water, CC, t, Tank} \rangle + 30\langle \text{Reserves, CC, t, Tank} \rangle$$

(期中フロー) これは常にダブルエントリーで流入、流出を分類して記述

1) インフロー、アウトフロー：これは水が地中から湧き出したり、消える事を示す。

$$\Delta \text{InFlow}(t)=10\langle \text{Water, CC, t, Tank} \rangle + 10\langle \text{InFlow, CC, t, Tank} \rangle \quad \text{流入=利益}$$

$$\Delta \text{OutFlow}(t)=5\langle \text{Water, CC, t, Tank} \rangle + 5\langle \text{OutFlow, CC, t, Tank} \rangle \quad \text{流出=損失}$$

2－1) 債務取引：ここでは他の系から水を借りた取引と、それを返却した取引を示す。

$$\Delta \text{Debt_In}(t)=15\langle \text{Water, CC, t, Subject} \rangle + 15\langle \text{Water_Liabilities, CC, t, Subject} \rangle \quad \text{水を借りる}$$

$$\Delta \text{Debt_Out}(t)=5\langle \text{Water, CC, t, Subject} \rangle + 5\langle \text{Water_Liabilities, CC, t, Subject} \rangle \quad \text{水を返す}$$

2－1) 債権取引：ここでは他の系へ水を貸した取引と、それが返却された取引を示す。

$$\Delta \text{Credit_In}(t)=30\langle \text{Water, CC, t, Subject} \rangle + 30\langle \text{Water_Credit, CC, t, Subject} \rangle \quad \text{水を貸す}$$

水が出て行っているが、損失ではなく、貸水である。

$$\Delta \text{Credit_Out}(t)=5\langle \text{Water, CC, t, Subject} \rangle + 5\langle \text{Water_Credit, CC, t, Subject} \rangle$$

水が入っているが、利益ではなく返却である。利子水を利益で記述することもできる。

ダブルエントリーにすることで、単なる水の変化だけでなく水に対する経済取引が可能となる！！！！

(残高試算表) これは期中フローの残高 (相殺) 計算

$$\begin{aligned}\text{TrialBalance}(t) &= \Delta \text{PS}(t) \equiv (\Delta \text{InFlow}(t) + \Delta \text{OutFlow}(t) + \Delta \text{Debt_In}(t) + \Delta \text{Debt_Out}(t) + \Delta \text{Credit_In}(t) \\ &+ \Delta \text{Credit_Out}(t)) \\ &= 10^{\langle \text{Water}, \text{CC}, t, \text{Tank} \rangle} + 10^{\langle \text{InFlow}, \text{CC}, t, \text{Tank} \rangle} + 5^{\langle \text{OutFlow}, \text{CC}, t, \text{Tank} \rangle} + 10^{\langle \text{Water_Liabilities}, \text{CC}, t, \\ &\text{Subject} \rangle} + 25^{\langle \text{Water_Credit}, \text{CC}, t, \text{Subject} \rangle}\end{aligned}$$

(振替変換) インフロー、アウトフローをリザーバに分類変え (振替)

$$\begin{aligned}\text{InFlowTransfer}(t) &= 10^{\langle \text{InFlow}, \text{CC}, t, \text{Tank} \rangle} + 10^{\langle \text{Reserves}, \text{CC}, t, \text{Tank} \rangle} \\ \text{OutFlowTransfer}(t) &= 5^{\langle \text{OutFlow}, \text{CC}, t, \text{Tank} \rangle} + 5^{\langle \text{Reserves}, \text{CC}, t, \text{Tank} \rangle}\end{aligned}$$

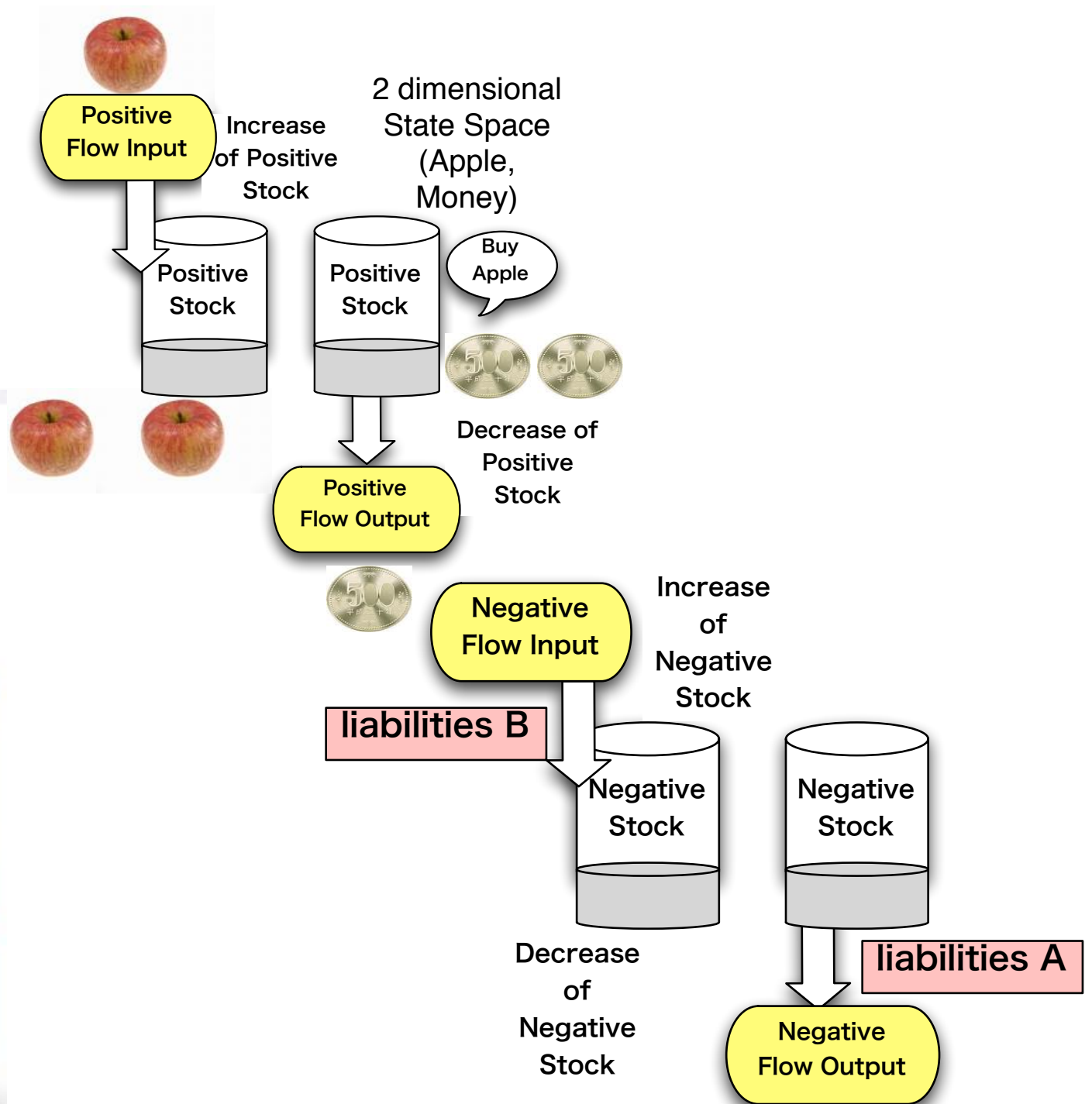
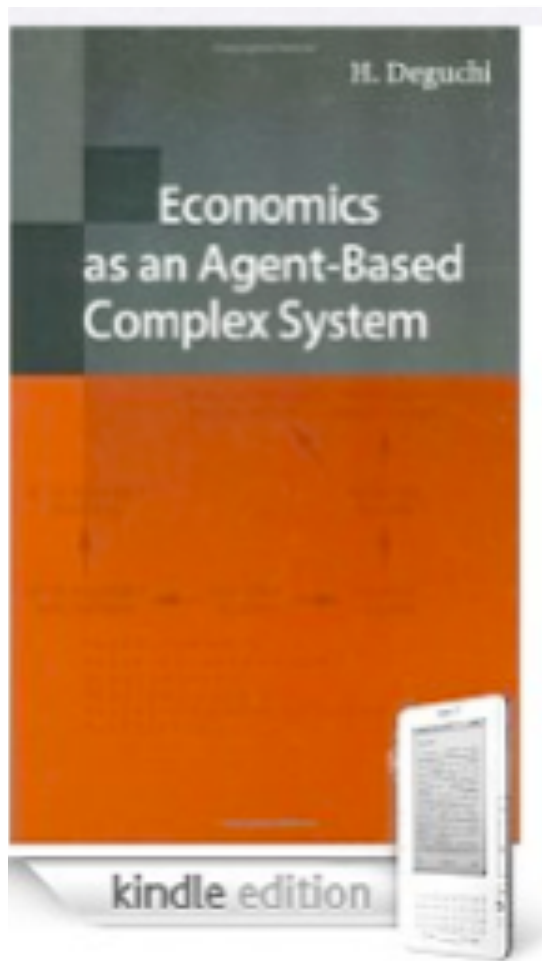
(期末残高) 期首の状態に残高を加えバー演算で状態の繰越計算をする

$$\begin{aligned}\text{PS}(t+1) &\equiv (\text{PS}(t) + \text{TrialBalance}(t) + \text{InFlowTransfer}(t) + \text{OutFlowTransfer}(t)) \\ &\equiv (30^{\langle \text{Water}, \text{CC}, t, \text{Tank} \rangle} + 10^{\langle \text{Water}, \text{CC}, t, \text{Tank} \rangle} + 10^{\langle \text{InFlow}, \text{CC}, t, \text{Tank} \rangle} + 5^{\langle \text{OutFlow}, \text{CC}, t, \text{Tank} \rangle} \\ &+ 10^{\langle \text{Water_Liabilities}, \text{CC}, t, \text{Subject} \rangle} + 25^{\langle \text{Water_Credit}, \text{CC}, t, \text{Subject} \rangle} \\ &+ 10^{\langle \text{InFlow}, \text{CC}, t, \text{Tank} \rangle} + 10^{\langle \text{Reserves}, \text{CC}, t, \text{Tank} \rangle} + 5^{\langle \text{OutFlow}, \text{CC}, t, \text{Tank} \rangle} + 5^{\langle \text{Reserves}, \text{CC}, t, \text{Tank} \rangle}) \\ &= 20^{\langle \text{Water}, \text{CC}, t, \text{Tank} \rangle} + 35^{\langle \text{Reserves}, \text{CC}, t, \text{Tank} \rangle} + 10^{\langle \text{Water_Liabilities}, \text{CC}, t, \text{Subject} \rangle} \\ &+ 25^{\langle \text{Water_Credit}, \text{CC}, t, \text{Subject} \rangle}\end{aligned}$$

注：ここでは水の現物は資産として20だがその内訳は、水の借りが10、水の貸しが25で、結局資産の内部留保は35と増えている。つまり初期に比べると水の現物が10減っているが、内部留保は5増えており、同時に水の負債が10、債権が25増えており、10の水資産の減少と、差引15の水債権の増加で、内部留保が5増えるという計算になっている。

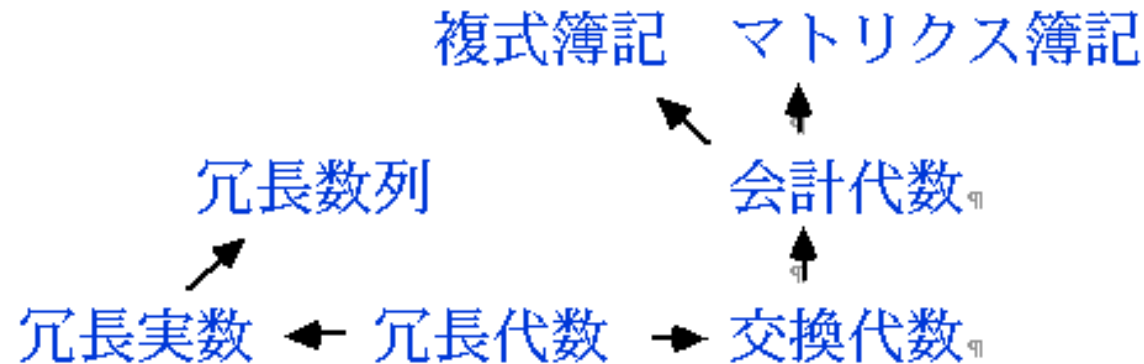
このようなマイナスのストック (負債) の増減や資産としての債権 (これはプラスのストック) によって状態変化が記述されるシステムの状態変化は、単純な入出力系のシステム記述に比べて遥かに複雑となっている。これがこの種のシステム記述で、複式での交換代数記述が必要とされる所以である。

交換代数の公理系による定式化



構造主義的公理アプローチ： 公理アプローチは何を狙うか！！

- 簿記という繰り返しのある表層構造の深層構造を定式化することが重要。ここでの理論はマテシッチ、井尻の理論を継承する形で、会計学の公理系を、簿記を抽象化する代数系からスタートしている。



- 簿記の深層構造の抽象化の諸関係

オークルスト、マテシッチ、井尻のアプローチ

- ミクロ、マクロ会計の公理化の試みは、いずれも集合論以降の公理化の系譜となるが、もとの取引の代数的な構造というよりは、取引の満たすべき条件を、列記する公準的な記述になっている。
- 1) オークルストの公理系の問題
 - オークルストの理論は、ミクロ会計にも通用するという。いわゆる会計公準に近い構成になっている。
- 2) 井尻の議論：会計を電気回路に例え多元簿記を目指したが、公理化には成功していない。
- 3) マテシッチの公理の整理：
 - 一方で簿記の抽象化を進め、マトリクス簿記を展開し、それを経済学に結び付けようとしたが、ストーンの弟子のパヤットはマテシッチを知らず、マテシッチもパヤットを知らなかった！！！！

状態空間複雑性と交換代数

経済システムの状態複雑性の基礎を与える作業

- * 簿記は経済的エージェントの状態記述を与える。簿記のシステムは冗長代数と交換代数という二つの公理系で特徴付けられる。
- * 借方、貸方は同値類として特徴付けられ、その同値類が二つしかないことに対応する。
- * 左右のバランスなど簿記であたりまえと思っていた性質は、我々の体系で公理から証明される定理として示される。
- * お金以外の単位で財を計る多元簿記も我々の代数系で扱える。
- * 500円の林檎を現金で売るという取引は、“500現金 \Leftrightarrow 500 $\hat{}$ 林檎”で表わされる。今 a を勘定科目とすると、 \hat{a} は勘定科目 a の減少を意味する。 $\hat{}$ 林檎 は、林檎の減少、即ち林檎を売るという取引を記述するための勘定科目となる。

経済交換の公理的基礎：交換関係の公理

- (1) $\forall a, b \quad a \Leftrightarrow b \equiv \hat{a} \Leftrightarrow \hat{b}$
- (2) $\forall a, b, c \quad a \Leftrightarrow b \text{ and } b \Leftrightarrow c \rightarrow \neg(a \Leftrightarrow c)$
- (3) $\forall a, b \quad a \Leftrightarrow b \equiv b \Leftrightarrow a$
- (4) $\forall a, b \quad a \Leftrightarrow b \rightarrow \neg(a \Leftrightarrow \hat{b})$
- (5) $\forall a, b, c \quad \neg(a \Leftrightarrow b) \text{ and } \neg(b \Leftrightarrow c) \rightarrow \neg(a \Leftrightarrow c)$
- (6) $\forall a \exists b \quad a \Leftrightarrow b$
- 交換関係を表わすために我々は、拡大基底集合 Γ 上の二項関係 \Leftrightarrow に対して、これら6つの公理は、“意味があり認められる取引の記述”の持つ内的拘束を特徴付けている。
- (1) $\forall a, b \quad a \Leftrightarrow b \equiv \hat{a} \Leftrightarrow \hat{b}$
 - a =現金, b = $\hat{}$ 林檎とし, c =売り掛け金, とすると公理(1)は林檎を売り現金を入手する取引($a \Leftrightarrow b$)が取引関係を満たした意味がある取引ならば, 現金を払って林檎を購入する取引($\hat{a} \Leftrightarrow \hat{b}$, ただし $\hat{\hat{b}} = \hat{\hat{}} \text{ 林檎} = \text{林檎}$)もやはり, 取引関係を満たした, 認められる取引記述であることを意味している。

交換関係の公理の性質

- (2) $\forall a, b, c \quad a \Leftrightarrow b \text{ and } b \Leftrightarrow c \rightarrow \neg(a \Leftrightarrow c)$
 - 公理(2)は、林檎を売り現金を取得する取引と、林檎を売り、売掛金を取得する取引が、取引関係を満たす取引ならば、現金と売掛金を同時に得るという取引は、取引関係の公理を満たさず、認められない取引記述であることを主張している。
- (3) $\forall a, b \quad a \Leftrightarrow b \equiv b \Leftrightarrow a$
 - 公理(3)は、取引関係の反射率。
- (4) $\forall a, b \quad a \Leftrightarrow b \rightarrow \neg(a \hat{\Leftrightarrow} b)$
 - 公理(4)は、林檎を売って現金を得るという取引が、取引関係を満たす取引であるならば、林檎と現金を同時に取得するという取引は取引関係の公理を満たさないことを主張する。
- (5) $\forall a, b, c \quad \neg(a \Leftrightarrow b) \text{ and } \neg(b \Leftrightarrow c) \rightarrow \neg(a \Leftrightarrow c)$
 - 公理(5)は、取引関係にないという関係($\neg \Leftrightarrow$)が、推移則を満たすことを主張している。

取引不可能関係と二つの同値類

- 【定義】 取引不可能関係 ▲
 - 任意の $a, b \in \Gamma$ に対して, a と b が取引不可能関係に有るとは, $\neg(a \Leftrightarrow b)$ が成立することである。このときこの関係を, $a \blacktriangle b$ と記すことにする。そしてこの Γ 上の二項関係 \blacktriangle を取引不可能関係と呼ぶ。
- 【定理】
 - 取引不可能関係 \blacktriangle は, Γ の上の同値関係であり, Γ はこの同値関係によって唯二つの同値類に分割される。

勘定科目と借方・貸方の差

- $\Lambda = \{\text{資産}, \text{負債}, \text{資本}, \text{利益}, \text{コスト}\}$
- $\hat{\Lambda} = \{\hat{\text{資産}}, \hat{\text{負債}}, \hat{\text{資本}}, \hat{\text{利益}}, \hat{\text{コスト}}\}$
- 借り方 = $\{\text{資産}, \hat{\text{負債}}, \hat{\text{資本}}, \hat{\text{利益}}, \text{コスト}\}$
- 貸し方 = $\{\hat{\text{資産}}, \text{負債}, \text{資本}, \text{利益}, \hat{\text{コスト}}\}$
- $\Gamma = \Lambda \cup \hat{\Lambda} = \text{借り方} \cup \text{貸し方}$
 - ここで Λ と $\hat{\Lambda}$ の分割と、借り方と貸し方の分割が異なることがポイント
- 【命題】 任意の会計ベクトル $x \in [\Gamma]$ は、借り方ベクトルと貸し方ベクトルの和に一意に分解できる。

冗長代数 Ω ： $\hat{}$ と $\overline{}$ を含む演算体系の公理的定式化

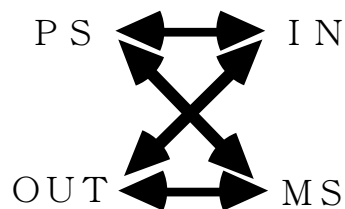
- 冗長代数 Ω : Ω の任意の元に対して二項演算 $+$ と単項演算 $\hat{}$ と $\overline{}$ 、スカラー倍 ax が定義され以下の公理を満たす。
- (1) $\forall x, y \in \Omega \quad x + y = y + x$ (2) $\forall x, y, z \in \Omega \quad (x + y) + z = x + (y + z)$
- (3) $\forall x \in \Omega \quad x + 0 = x$, (4) $\forall a, b \in T, \forall x \in \Omega \quad a(bx) = (ab)x$
- (5) $\forall x \in \Omega \quad 1x = x, \quad 0x = 0$ (6) $\forall a, b \in T, \forall x, y \in \Omega \quad (a + b)x = ax + bx$
- (7) $\forall a \in T, \forall x, y \in \Omega \quad a(x + y) = ax + ay$
- (8) $\forall x \in \Omega \quad \overline{\overline{x}} = x$, (9) $\forall x \in \Omega \quad \hat{\hat{x}} = x$
- (10) $\forall x, y \in \Omega \quad \overline{(x + y)} = \overline{\overline{x}} + \overline{\overline{y}}$
- (11) $\forall x, y \in \Omega \quad \hat{(x + y)} = \hat{x} + \hat{y}$
- (12) $\forall x, y \in \Omega \quad \overline{(\hat{x} + y)} = 0 \equiv \overline{x} = \overline{y}$ (13) $\forall x \in \Omega \quad \overline{(\hat{x})} = \hat{(\overline{x})}$
- (14) $\forall a \in T, \forall x \in \Omega \quad \overline{(ax)} = a(\overline{x}), \quad \hat{(ax)} = a(\hat{x})$
- (15) $\forall x, y \in \Omega \quad x + y = 0 \rightarrow x = 0 \wedge y = 0$ (16) $\forall z, x \in \Omega \quad x + z = x \rightarrow z = 0$
- (17) $\forall a \in T, \forall x \in \Omega \quad ax = 0 \rightarrow a = 0 \vee x = 0$

冗長代数の例：冗長実数

- 【例】 冗長実数
- $R_r = \{(x_1, x_2) \mid x_1, x_2 \in \mathbb{R}_+\}$
- $(x_1, x_2) + (y_1, y_2) = (x_1 + x_2, y_1 + y_2)$
- $a(x_1, x_2) = (ax_1, ax_2), a \in \mathbb{R}_+$
- $\hat{}(x_1, x_2) = (x_2, x_1)$
- $\overline{}(x_1, x_2) = (x_1 - x_2, 0)$ if $x_1 - x_2 \geq 0$
- $\phantom{\overline{}}(x_1, x_2) = (0, x_2 - x_1)$ if $x_2 - x_1 > 0$
- 注意:これが冗長という理由は、たとえば普通の実数で-1に当たる元が、 $(4, 5), (1, 2), (8, 9)$ と複数存在するからである。単項演算子 $\overline{}$ は、この冗長度を除去する働きを持つ。たとえば、 $\overline{}(4, 5) = (0, 1)$ 、 $\overline{}(8, 9) = (0, 1)$ となる。

冗長代数から交換代数の拡張構成定理 1

- 今 Λ を冗長代数 Ω の基本基底集合, \Leftrightarrow を Λ 上の二項関係で, 交換関係の公理のうちで, (2),(3),(5),(6)を満たすものであるとする。このとき, 関係 \Leftrightarrow は, 拡大基底 Γ 上の交換関係に一意に拡張される。
- (2) $\forall a,b,c \quad a \Leftrightarrow b \text{ and } b \Leftrightarrow c \rightarrow \neg(a \Leftrightarrow c)$
- (3) $\forall a,b \quad a \Leftrightarrow b \equiv b \Leftrightarrow a$
- (5) $\forall a,b,c \quad \neg(a \Leftrightarrow b) \text{ and } \neg(b \Leftrightarrow c) \rightarrow \neg(a \Leftrightarrow c)$
- (6) $\forall a \exists b \quad a \Leftrightarrow b$



拡張構成定理 2

拡張の意味：マイナスのストックの有る

インプットアウトプットシステム

- $\Lambda' = \{P S, I N, M S, O U T\}$
- Λ から Λ' への全射, $g: \Lambda \rightarrow \Lambda'$ が存在して、 Λ 上の交換関係 \Leftrightarrow が次のように入るとき、関係 \Leftrightarrow は Γ 上の交換関係の公理を満たす関係へ一意に拡張される。
 - (1) If $g(x) = P S$ and $g(y) = I N$ then $x \Leftrightarrow y$ and $y \Leftrightarrow x$
 - (2) If $g(x) = P S$ and $g(y) = M S$ then $x \Leftrightarrow y$ and $y \Leftrightarrow x$
 - (3) If $g(x) = O U T$ and $g(y) = I N$ then $x \Leftrightarrow y$ and $y \Leftrightarrow x$
 - (4) If $g(x) = O U T$ and $g(y) = M S$ then $x \Leftrightarrow y$ and $y \Leftrightarrow x$

交換関係の公理の意味

* **公理(2)**は、林檎を売り現金を取得する取引と、林檎を売り、売掛金を取得する取引が、取引関係を満たす取引ならば、現金と売掛金を同時に得るという取引は、取引関係の公理を満たさず、認められない取引記述であることを主張している。

* **公理(3)**は、取引関係の**反射率**。

* **公理(4)**は、林檎を売って現金を得るという取引が、取引関係を満たす取引であるならば、林檎と現金を同時に取得するという取引は取引関係の公理を満たさないことを主張する。

* **公理(5)**は、取引関係にないという**関係($\neg \Leftrightarrow$)**が、推移則を満たすことを主張している。

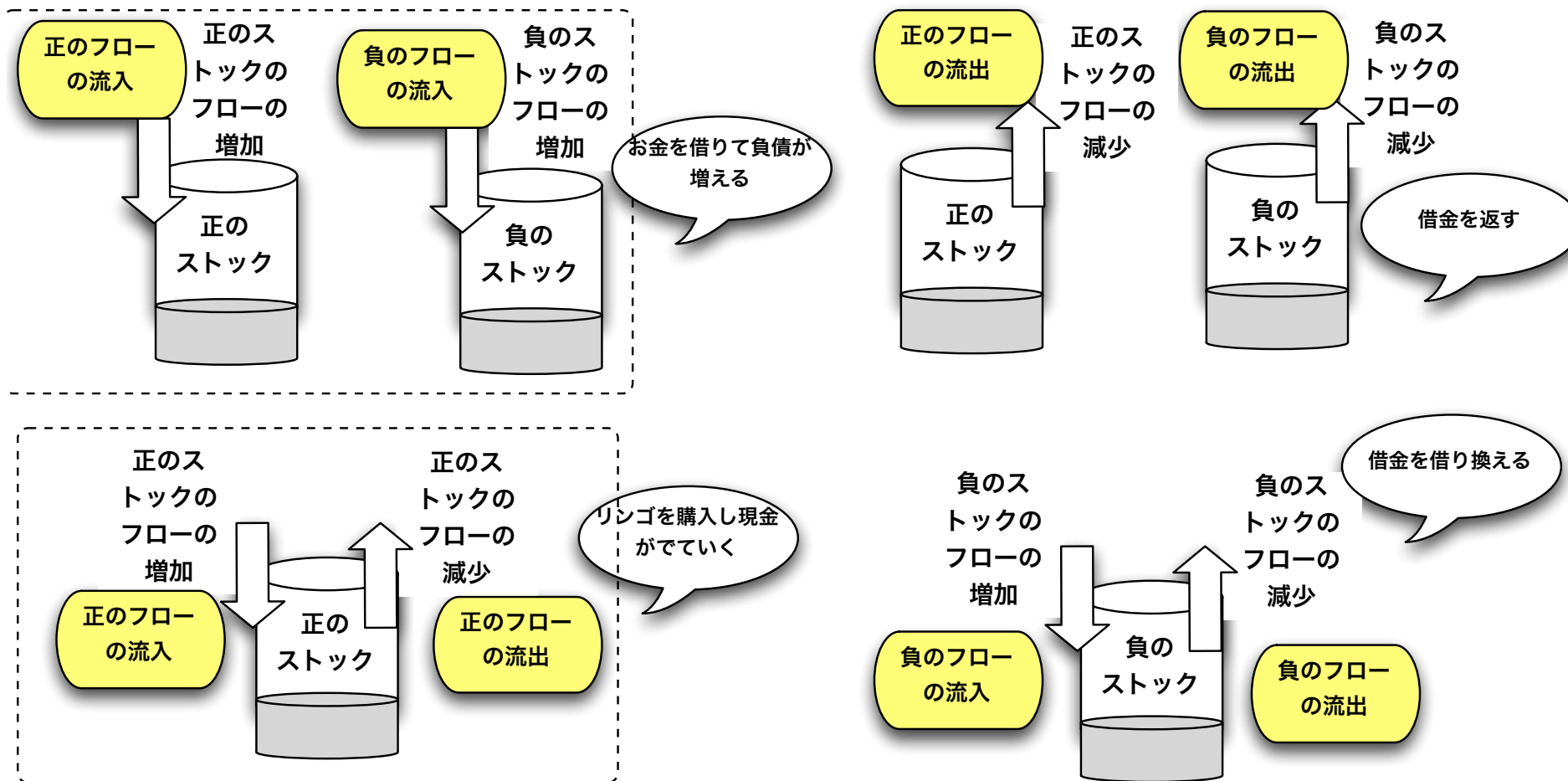
【定義 5. 2. 1 4】 **取引不可能関係 \blacktriangle**

任意の $a, b \in \Gamma$ に対して、 a と b が取引不可能関係に有るとは、 $\neg(a \Leftrightarrow b)$ が成立することである。このときこの関係を、 $a \blacktriangle b$ と記すことにする。そしてこの Γ 上の二項関係 \blacktriangle を取引不可能関係と呼ぶ。

【定理 5. 2. 1】

取引不可能関係 \blacktriangle は、 Γ の上の同値関係であり、 Γ はこの同値関係によって唯二つの同値類に分割される。

経済のシステムでは一方的に正や負のストックが増えるのではなく現金が増えて負債が増えるように取引という形で状態の変化が生じるパターンが多い（簿記的記述）、それを含め非常に多種類に分類された財や負債の状態とその変化を記述する必要がある、社会会計ではある。そこにはさらに分類の間のアグリゲーションや按分変換などの操作が求められる。これらを与える代数系が交換代数で、それをそのままシステム化したものがAADL



交換代数のソフトウェア実装

交換代数のソフトウェア実装

代数的仕様記述による、会計的なデータ記述に
対する現場レベルでのモジュール記述と絶えず
変化するビジネスプロセスに対応できるモデル
開発

＊システム分析・上流設計からソリューションまでの連結

＊毎年変化するシステムに耐えられるロバストな情報システム
設計

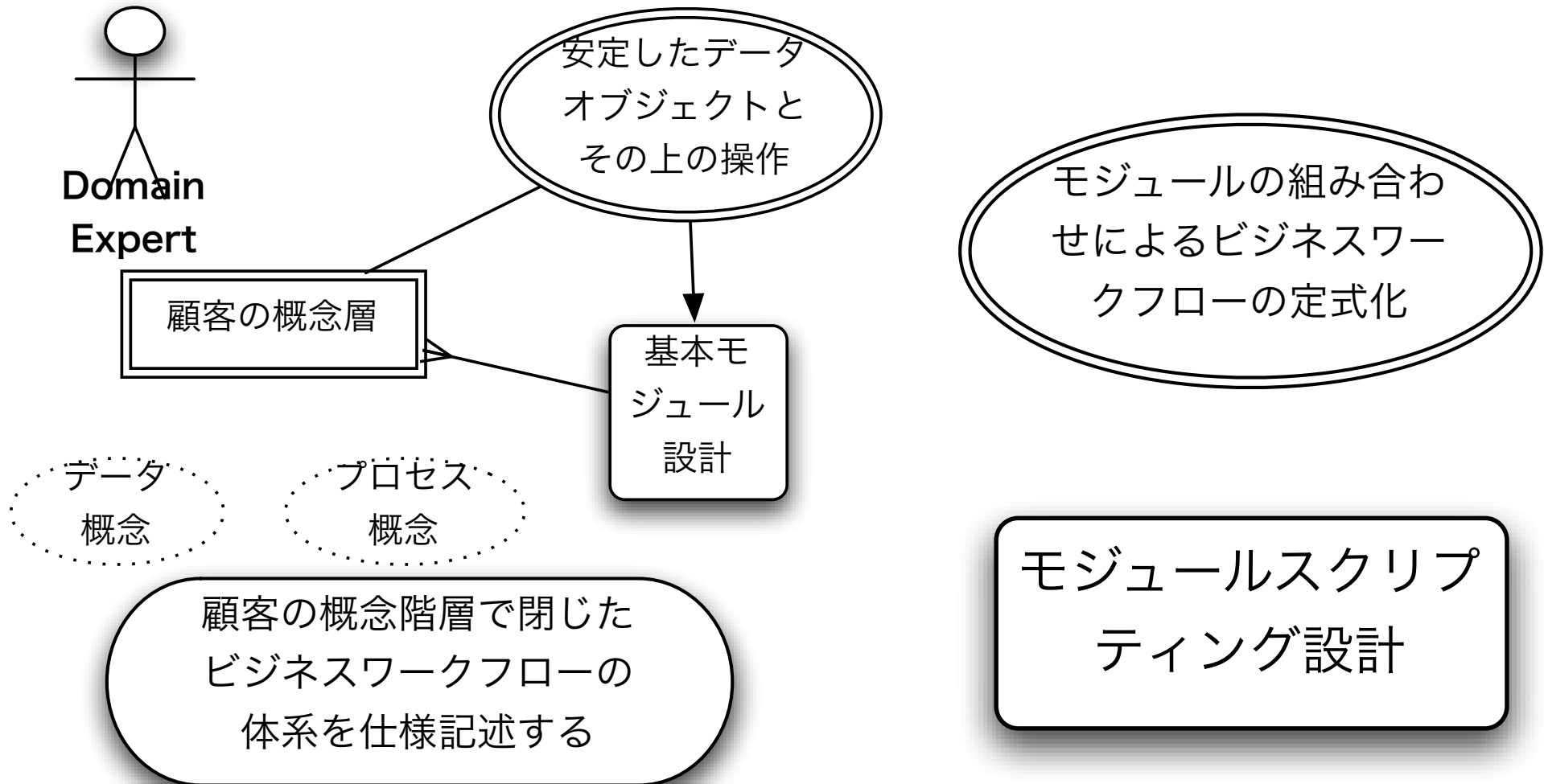
大規模なモジュールの開発管理

- 長期的なモジュールのライフサイクル管理
 - 現場レベルでのモジュールのメンテナンス→”改善”可能なモジュール管理の必要性！！
- 数千から数万のモジュールの開発管理：DSLとしてのAADLの開発
 - 仕様記述の統一と、上位のサービス層でのモデル化仕様の規格化の重要性！ そこで代数的な社会会計表現に基づいた集合論的な仕様記述を基盤にする。これは本来の現場のデータ構造の層！
 - その上で、集合論的な仕様記述のレベルにあったDSL(Domain Specific Language)によるモジュールの仕様記述と、モジュールのコンパイルが必要。そのためにAADL(Algebraic Accounting Description Language)を開発した。
 - これは仕様から実装モジュールに展開するときの「ぶれ」を防ぐため！！

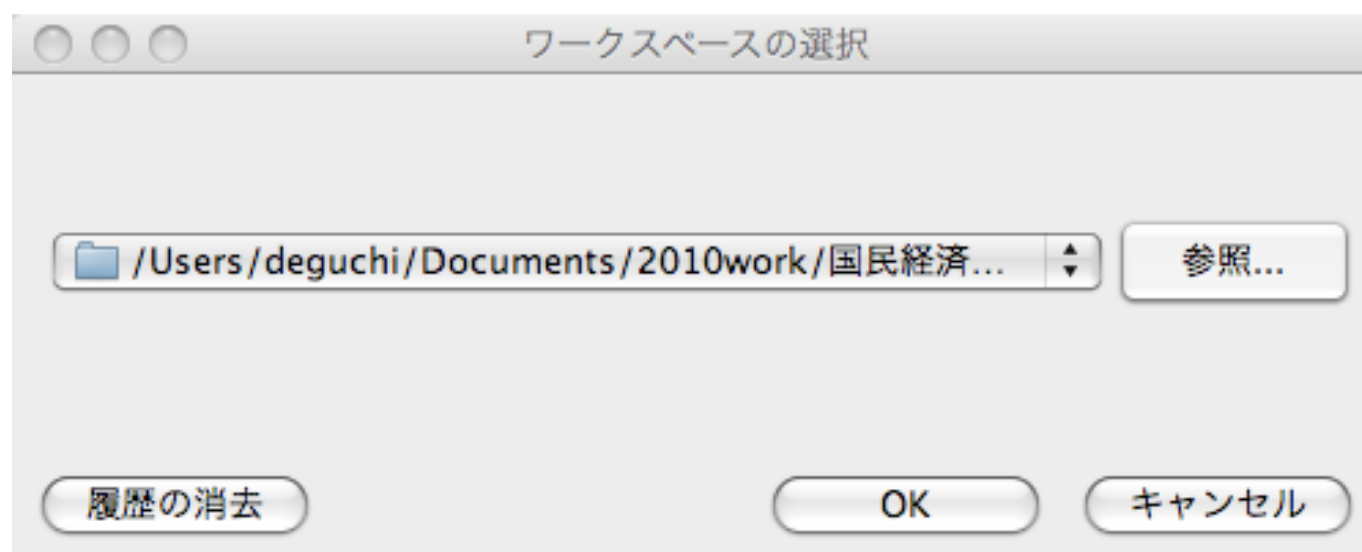
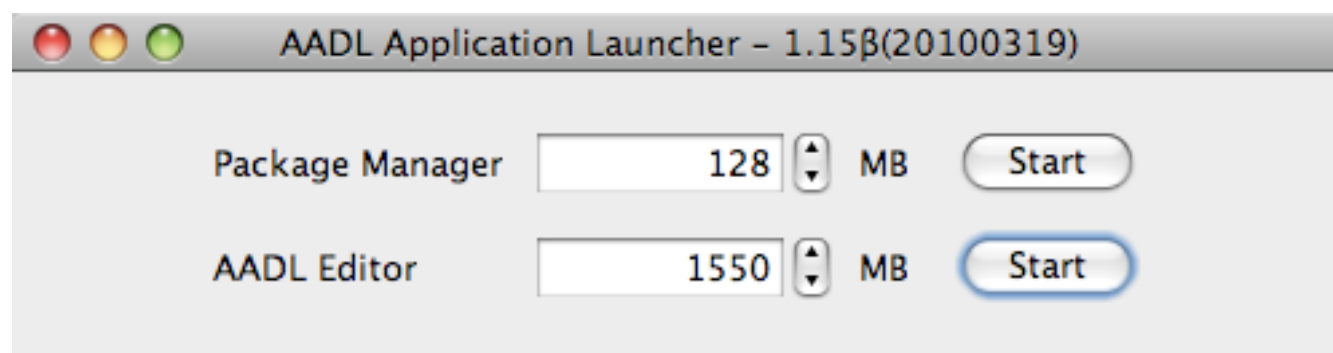
仕様記述言語としてのAADLの必要性

- (1) 交換代数のパッケージを中心に、仕様記述の代数的集合的方法を標準化し、そのパッケージを用いて個々のモジュールを実装する。
 - モジュールの組み合わせは、ビジュアルなスクリプトで行う。
 - 課題：仕様記述と実装の間に乖離が生じる。開発が仕様記述チームと実装チームで二極化する。実装がテストデータから逆設計されやすい。（一般に仕様と実装の乖離が生じ、モジュールの上位設計の方が難しい）
- (2) 仕様記述の代数的集合論的方法を標準化した上で、その水準にあったDSL(AADL)を提供し、AADLでのモジュール記述を現場とシステムエンジニアがペアプロするアジャイルプログラミングスタイル。
 - AADLが仕様記述のレベルに対応し十分な機能を持てば、長期的メンテが現場レベルで可能。

サービス指向の階層アーキテクチャ

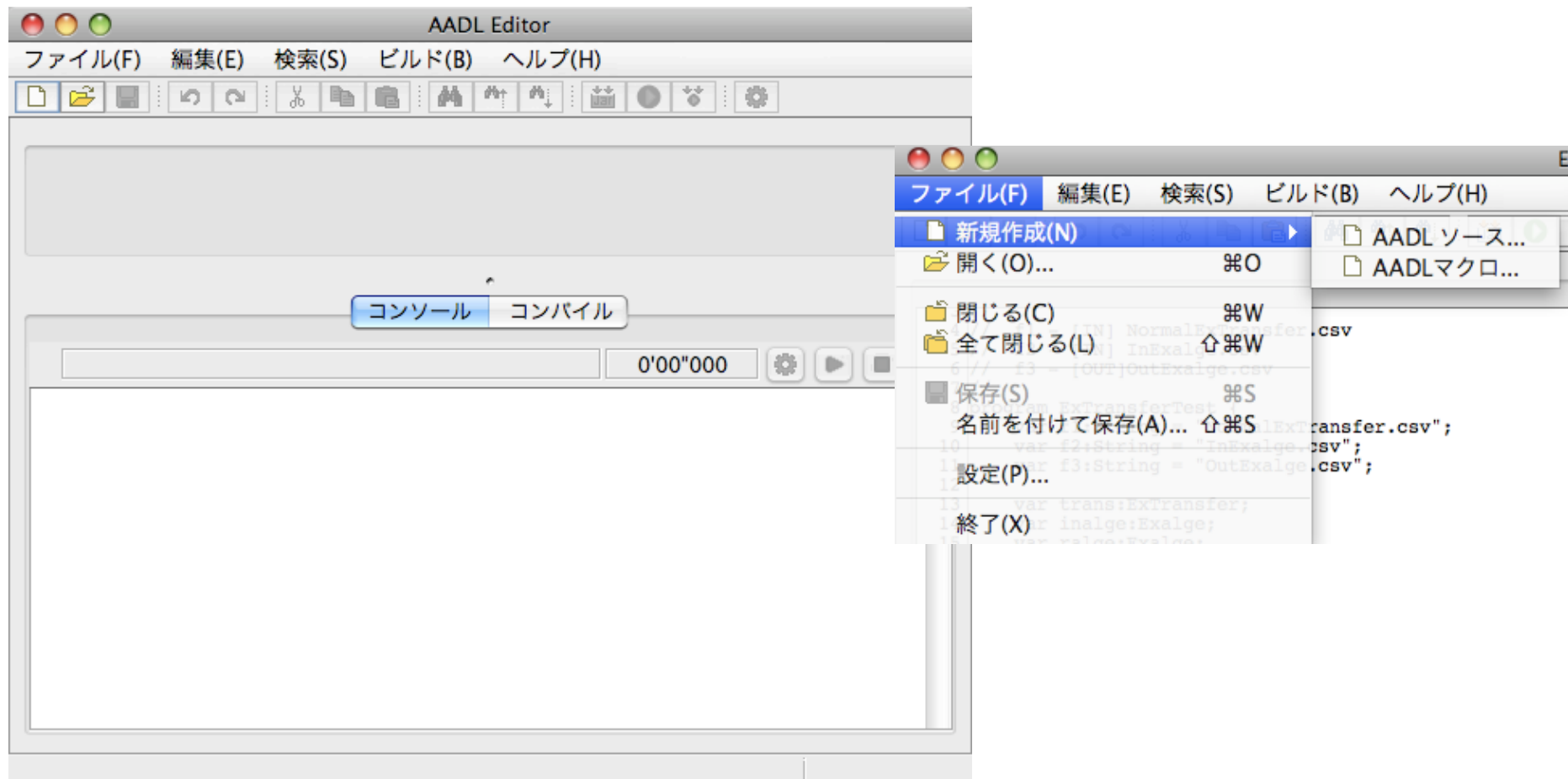
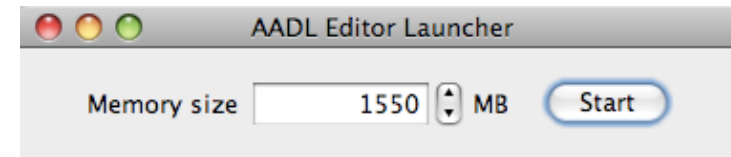


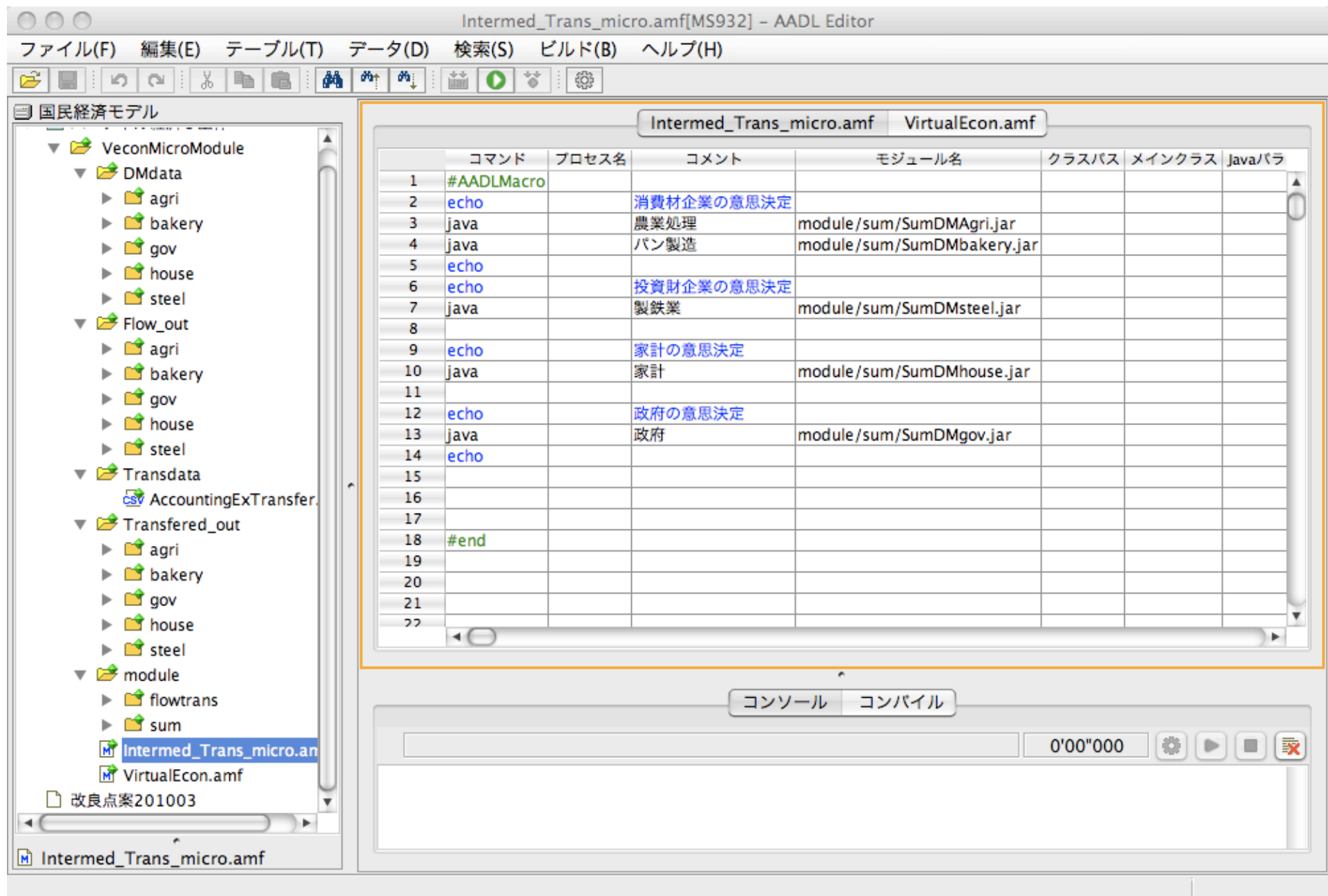
- 代数的仕様記述を採用：交換代数に基づく集合論的仕様記述



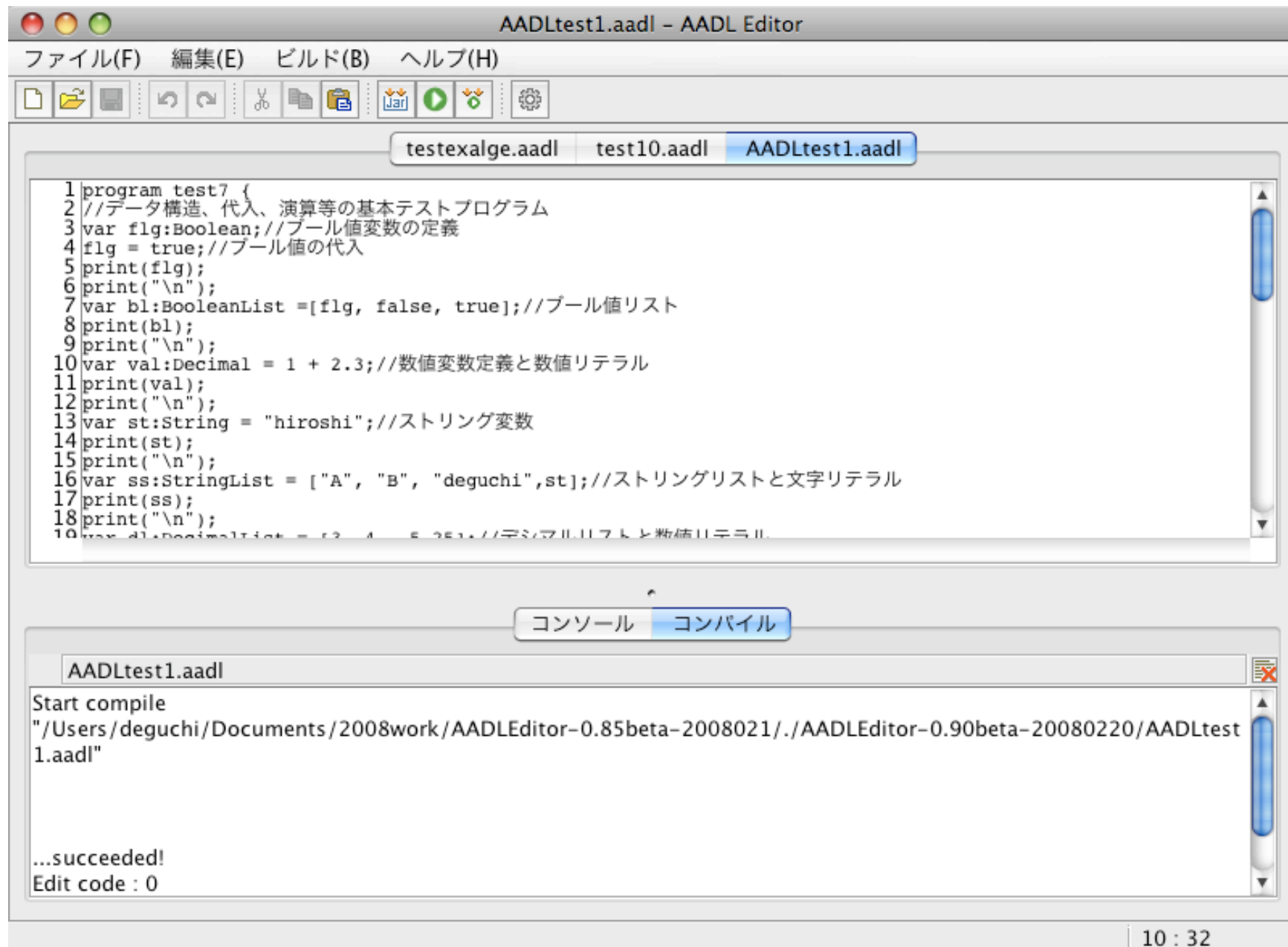
AADL言語の特色

- 交換代数に基づいた演算体系を支援し、社会統計や社会会計の計算を行うための「領域固有言語」である。





AADLエディタ（開発環境）



モジュール設計の基本的な考え方

- データの交換代数表記
 - 一つの業務処理のまとまりからなるオブジェクト
 - 上記の最小単位のモジュールは一般に下記のように交換代数で表記される。

- (1) 4種類の基本基底

項目(name)、単位(unit)、時間(time)、主体(subject)
「誰が、何を、いつ、どれだけ処理したデータか」を管理

$$\Lambda^{name}, \Lambda^{unit}, \Lambda^{time}, \Lambda^{subject}$$

- (2) 交換代数ベクトル

$$x = \sum \left\{ \begin{array}{l} x(e, u, t, s) \langle e, u, t, s \rangle \\ \left| e \in \Lambda^{name}, u \in \Lambda^{unit}, t \in \Lambda^{time}, s \in \Lambda^{subject} \right. \end{array} \right\}^8$$

AADL記述例：ラスパイレス固定デフレータ



$$LD = \frac{\sum_{e \in \Lambda^{name}} P_e^t Q_e^0}{\sum_{e \in \Lambda^{name}} P_e^0 Q_e^0}$$

$$LD = \frac{\sum \left\{ \begin{array}{l} p(e,t)q(e,t_0) | \\ e \in \Lambda^{name}, \\ p(e,t) = \|\text{Proj}[\langle e, PI, t, \# \rangle](Xp)\|, q(e,t_0) = \|\text{Proj}[\langle x, VQ, t_0, \# \rangle](Xq)\| \end{array} \right\}}{\sum \left\{ \begin{array}{l} p(e,t_0)q(e,t_0) | \\ e \in \Lambda^{name}, \\ p(e,t_0) = \|\text{Proj}[\langle e, PI, t_0, \# \rangle](Xp)\|, q(x,t_0) = \|\text{Proj}[\langle x, VQ, t_0, \# \rangle](Xq)\| \end{array} \right\}}$$

AADL記述例：ラスパイレス固定デフレータ

```
var numerator:DecimalList  
= { na*nb | e <- subPI,  
    a = proj[<e,"PI",time, "#">](algePI),  
    b = proj[<e,"VQ",refTime, "#">](algeVQ),  
    na = norm(a), nb = norm(b),  
    !isEmpty(a) && !isEmpty(b) && na != 0 && nb != 0  
};  
var numeratorSum:Decimal = sum(numerator);  
  
var denominator:DecimalList  
= { nc*nd | e <- subPI,  
    c = proj[<e,"PI",refTime, "#">](algePI),  
    d = proj[<e,"VQ",refTime, "#">](algeVQ),  
    nc = norm(c), nd = norm(d),  
    !isEmpty(c) && !isEmpty(d) && nc != 0 && nd != 0  
};  
var denominatorSum:Decimal = sum(denominator);  
  
var LD: Decimal = sum(numerator) / sum(denominator);
```

AADLの基本文法

AADLの基本文法

- program プログラム名 { 文 }
- AADLの文は、式の組み合わせにより構成され、単一の文はセミコロンで終わる必要があります。複数の文は {} で囲まれたブロックに記述することが出来ます。

AADLで利用できるデータ型

- Boolean 真偽値
- Decimal 数値 (実数)
- String 文字列
- ExBase 交換代数の一つの基底
- Exalge 交換代数の元 (元は、交換代数： $3\langle\text{orange}\rangle + 2\langle\text{apple}\rangle + 5\langle\text{banana}\rangle$ などを表す)
- TransTable 交換代数の振替変換テーブル
- TransMatrix 交換代数の按分変換マトリックス
- BooleanList 複数のBooleanのリスト
- DecimalList 複数のDecimalのリスト
- StringList 複数のStringのリスト
- ExBaseSet 交換代数基底 (ExBase) の集合
- ExAlgeSet 交換代数の元 (Exalge) の集合

利用可能なリテラル 1

- 真偽値リテラル (Boolean型) 「true」 (真)、 「false」 (偽)
- 数値リテラル (Decimal型) 「3」、 「-1.25」 などの整数、小数値
- 文字列リテラル (String型) 「"ABC"」 など、ダブルクォーテーションで囲まれたもの。以下のエスケープシーケンスも利用可能。
 - `¥b` バックスペース `¥t` タブ `¥n` 改行 `¥r` 復帰
 - `¥f` 改ページ `¥'` シングルクォート `¥"` ダブルクォート
 - `¥¥` `¥文字` `¥nnn` 8進数表記 (nは0から7)
 - `¥unnnn` ユニコード表記 (nは0から9、aからf、AからF)

利用可能なリテラル 2

- 交換代数基底リテラル (ExBase型)
「<"apple","yen","Y1990","subject">」など、<>で囲まれたもの。<>内には、文字列リテラル、String型変数、String型の値を返す関数等を指定できる。<>の並びは、名前キー、単位キー、時間キー、主体キーの順で、名前キー以外は省略可能。
- 交換代数リテラル (Exalge型) '@'演算子により、数値オブジェクト (Decimal) と交換代数基底オブジェクト (ExBase) から交換代数オブジェクト (Exalge) を生成できる。「3.01@<"apple","yen">」のように記述し、交換代数オブジェクトを記述できる。
- リストリテラル：[] でオブジェクトを（カンマ区切りで）複数指定することで、リストを生成できます。リストの要素に指定可能なデータ型は、Boolean、Decimal、String、ExBase、Exalgeのみ。また、要素に異なるデータ型のオブジェクトを含めることは出来ない。

AADL Sample

- test002 {
 - ExAlgeSet X <<- csvFile("hoge.csv"); // read CSV
 - ExBaseSet E <<- csvFile("base.csv"); /* read CSV */
 - Exalge a1 = 1<Yamada> + 2^<Yamada, ¥, Y2000>;
 - Exalge a2 = 3.1<Tanaka> + 3.2^<Tanaka> * 3;
 - Exalge a3 = a1 + ^a2;
 - ExBaseSet e = [<Yamada>, <Tanaka>];
 - ExBaseSet e2 = E %union% [<Yamada>, <Tanaka>];
 - ExBaseSet e3 = e %intersection% e2;
 - ExBaseSet e4 = e %difference% e2;
 - Decimal d1 = | proj[E](X) |;
 - Exalge a5 = sum(X);
 - ExAlgeSet s1 = { proj[e](x) | x <- X, |x| > 5 };
 - s1 ->> csvFile("outHoge.csv", "UTF-8");
 - }

AADL予約語と変数

- 予約語一覧

AADL	aadl	program	function
Boolean	Decimal	String	ExBase
Exalge	TransTable	TransMatrix	BooleanList
DecimalList	StringList	ExBaseSet	ExAlgeSet
csvFile	xmlFile	true	false
var	return	if	else
null	void	aadlRun	

- 変数： 変数の宣言は“var”キーワードを使用し、「var 変数名：データ型」の形式で記述する。AADLでは、一部を除き、使用する変数は変数を利用する前に宣言する必要がある。変数宣言では、必ずデータ型を指定する必要がある。
- 変数宣言では、値や式を直接代入することも可能。
- 例) var flg:Boolean;
 - var val:Decimal = 1 + 2;
 - var ss:StringLiteral = ["A", "B", "C"];
 - var alge:Exalge = 3.0@<"apple"> + 3.0@^<"cach">;

プログラム引数

- AADLでは、プログラム引数を取得するための文字列オブジェクトとなる特殊な定数が定義されています。「\$1」から「\$9」までの定数が利用でき、それぞれが文字列オブジェクトとなっています。この定数とプログラム引数との対応は、次の通りです。
 - \$1 1番目のプログラム引数
 - \$2 2番目のプログラム引数
 -
 - \$8 8番目のプログラム引数
 - \$9 9番目のプログラム引数

算術演算子一覧

- 算術演算子
- AADLで利用可能な算術演算子は、次の通り。基本的に数値オブジェクトにのみ利用できる。

演算子	式	演 算	利用可能データ型
-	$\cdot x$	符号反転	Decimal
+	$x + y$	加算 (文字列の場合は連結)	Decimal + Decimal、 String + String、 Exalge + Exalge
-	$x \cdot y$	減算	Decimal · Decimal
*	$x * y$	乗算	Decimal * Decimal、 Decimal * Exalge、 Exalge * Decimal
/	x / y	除算	Decimal / Decimal、 Exalge / Decimal
%	$x \% y$	剰余	Decimal % Decimal

比較演算子と論理演算子

演算子	式	演算
<code>==</code>	$x == y$	等しい
<code>!=</code>	$x != y$	等しくない
<code><</code>	$x < y$	x が y より小さい
<code>></code>	$x > y$	x が y より大きい
<code><=</code>	$x <= y$	x が y 以下
<code>>=</code>	$x >= y$	x が y 以上

演算子	式	演 算
<code>!</code>	$!x$	否定
<code> </code>	$x y$	x と y が共に真であれば、真
<code>&&</code>	$x \&\& y$	x と y のどちらかが真であれば、真

特殊演算子と演算子の優先順位

演算子	式	演 算	利用可能データ型
\wedge	$\wedge x$	ハット 演算	ExBase、Exalge
\sim	$\sim x$	バー演算	Exalge
@	$x @ y$	Exalge 生成	Decimal @ ExBase、 ExBase @ Decimal
%union%	$x \%union\% y$	和集合	ExBaseSet、ExAlgeSet
%intersection%	$x \%intersection\% y$	積集合	ExBaseSet、ExAlgeSet
%difference%	$x \%difference\% y$	差集合	ExBaseSet、ExAlgeSet

·(単項)	高
!、~、^	↑
*、/、%、@	
+(二項)、·(二項)	
%union%、%intersection%、%defference%	
<、>、<=、>=	↓
==、!=	
&&	
	低

交換代数標準形でのファイル入出力

- 交換代数の標準形ファイルフォーマットに準じ、ファイルの入出力を記述することが出来ます。ファイル入出力に対応するデータ型は、`Exalge`、`ExAlgeSet`、`ExBaseSet`、`TransTable`、`TransMatrix`となります。特に`TransTable`、`TransMatrix`は、ファイルからの入力のみを想定したデータオブジェクト
- ファイル入出力には、ファイルの種類を示すキーワード（`"csvFile"`、`"xmlFile"`）と、ファイル入出力の方向を示す記号（`'->>'`、`'<<-'`）を組み合わせで記述する。ファイル入力は、変数宣言と共に記述することが可能。

ファイルの入出力

- ファイル入力

- // \$1が示すファイル名からCSVファイル形式で、デフォルトのエンコーディングで入力: `var alge:Exalge; alge <<- csvFile($1);`
- // \$2が示すファイル名からCSVファイル形式で、UTF-8エンコーディングで入力: `var table:TransTable <<- csvFile($2, "UTF-8");`
- // 指定のファイル名からXMLファイル形式で入力:
 - `var exbaseset:ExBaseSet <<- xmlFile("baseset.xml");`

- ファイル出力

- // \$1が示すファイル名のファイルへCSVファイル形式で、デフォルトのエンコーディングで出力: `alge ->> csvFile($1);`
- // \$2が示すファイル名のファイルへCSVファイル形式で、UTF-8エンコーディングで出力: `table ->> csvFile($2, "UTF-8");`
- // 指定のファイル名のファイルへXMLファイル形式で出力
 - `exbaseset ->> xmlFile("baseset.xml");`

組み込み関数、ユーザ定義関数

- 関数名([引数 [,引数]...])関数の呼び出しは、AADLでは式として評価。変数への代入や演算子との組み合わせも可能。この場合、関数が値を返す必要がある。
- ユーザ定義関数：ユーザ独自の関数記述する場所は、“program”キーワードで記述されたプログラム・ブロックの外側。
 - function 関数名([引数名:引数データ型[,引数名:引数データ型]...])[:戻り値データ型]{ 文 } : 戻り値データ型を省略すると、値を返さない関数。戻り値を返す関数を定義した場合、“return”キーワードにより値を返す。

- 組み込み関数

書 式	機 能
<code>get[ExBase](Exalge):Decimal</code>	指定した基底の値を交換代数から取り出す。対応する基底が存在しない場合は 0 を返す。
<code>proj[ExBase](Exalge):Exalge</code>	指定した基底のみを含む交換代数オブジェクトを返す。対応する基底が存在しない場合、要素を持たない交換代数オブジェクトを返す。
<code>proj[ExBaseSet](Exalge):Exalge</code>	指定した基底集合のみを含む交換代数オブジェクトを返す。対応する基底が存在しない場合、要素を持たない交換代数オブジェクトを返す。
<code>isEmpty(Exalge):Boolean</code>	交換代数オブジェクトに要素が存在しない場合に true を返す。

組み込み関数

書 式	機 能
<code>isEmpty(<i>ExalgeSet</i>):Boolean</code>	交換代数集合に要素が存在しない場合に true を返す。
<code>isEmpty(<i>ExBaseSet</i>):Boolean</code>	交換代数基底集合に要素が存在しない場合に true を返す。
<code>isEmpty(<i>BooleanList</i>):Boolean</code>	真偽値リストに要素が存在しない場合に true を返す。
<code>isEmpty(<i>DecimalList</i>):Boolean</code>	数値リストに要素が存在しない場合に true を返す。
<code>isEmpty(<i>StringList</i>):Boolean</code>	文字列リストに要素が存在しない場合に true を返す。
<code>norm(<i>Exalge</i>):Decimal</code>	交換代数オブジェクトに含まれる全要素の値の総和を返す。要素が存在しない場合は 0 を返す。
<code>sum(<i>ExAlgeSet</i>):Exalge</code>	交換代数集合の総和となる交換代数オブジェクトを返す。
<code>sum(<i>DecimalList</i>):Decimal</code>	数値リストに含まれる全要素の総和を返す。
<code>aggreTransfer(<i>Exalge</i> , <i>TransTable</i>):Exalge</code>	指定された交換代数オブジェクトを指定の振替変換テーブルにより振替変換を行う。
<code>divideTransfer(<i>Exalge</i> , <i>TransMatrix</i>):Exalge</code>	指定された交換代数オブジェクトを指定の按分変換テーブルにより按分変換を行う。
<code>pow(<i>a:Decimal</i> , <i>b:Decimal</i>):Decimal</code>	a の b 乗を計算した結果を返す。
<code>sqrt(<i>Decimal</i>):Decimal</code>	指定された値の平方根を返す。
<code>abs(<i>Decimal</i>):Decimal</code>	指定された値の絶対値を返す。
<code>min(<i>a:Decimal</i> , <i>b:Decimal</i>):Decimal</code>	指定された値の小さい方の値を返す。
<code>max(<i>a:Decimal</i> , <i>b:Decimal</i>):Decimal</code>	指定された値の大きい方の値を返す。
<code>ceil(<i>Decimal</i>):Decimal</code>	正の無限大に近い整数を返す。
<code>floor(<i>Decimal</i>):Decimal</code>	負の無限大に近い整数を返す。

繰り返し処理のための内包記法

- 変数名 = { 内包式 | 条件 };
 - {} で囲まれたブロックが内包記法。内包記法を記述する場合、必ず変数に代入する必要があり、代入できるデータ型は 内包式 の結果を複数格納できるリストデータ型（DecimalListやExAlgeSet等）に限定。
 - 内包記法の 条件 に、イテレータ式、評価式（Boolean値を返す）、エイリアスを、カンマで区切るにより複数記述できる。内包記法の条件に記述した処理は、記述した順序で実行される。
 - 内包記法の内包式に、単一の式（関数や演算）を記述する。式の実行結果が値を返す必要がある。値を返さない式は記述できない。
- イテレータ式
 - 要素変数名 <- リストオブジェクト
 - 要素変数名には、任意の名前を指定。この名前で、リストオブジェクト内の要素にアクセスできる。この式は、リストオブジェクトから順次要素を取り出し、全ての要素に対して内包記法の内包式を繰り返し実行する。要素変数名は、内包式で利用できる。イテレータ式を複数記述した場合、二重、三重の繰り返し処理が実行されると考えてください。

- 評価式

- Boolean値を返す関数や、論理演算子や比較演算子による評価式を指定する。ここで指定された条件を満たす場合のみ、内包記法の内包式が実行される。

- エイリアス

- 内包記法で実行される式の結果を一時変数として利用することが可能。エイリアスは、次のように記述する。

- エイリアス名 = 式

- ここで指定したエイリアス名は、内包式で利用できる。

- 内包記法例

- `var xPCSet:ExAlgeSet =`
- `{ proj[<"自営業主数","人",t,i>](xData) | t<-lambdaTby, i<-lambdaInd };`
- `var xPC:Exalge = sum(xPCSet);`

if文

- AADLでは、条件判断文を記述できる。
 - if (評価式) 文 [else if (評価式) 文] [else 文]
- 評価式には、Boolean値を返す関数や、論理演算子や比較演算子による評価式を記述する。文には、セミコロンの終わる単一の文、もしくは {} で囲まれたブロックを記述する。

JAVAコードの埋め込み

- AADLでは、JAVAプログラムを直接記述するための記述子が2種類、1) AADLから生成されたJAVAコードの先頭に配置されるJAVAヘッダーブロックと、2) AADL内の式として記述できるJAVAアクションブロックがある。これらのブロック内の記述は、AADLの文法としては評価されず、JAVAプログラムのコンパイル時に評価されます。
- JAVAヘッダーブロック：JAVAヘッダーブロックは、AADLから生成されたJAVAコードの先頭に配置されるブロック。
 - `@header{ JAVAコード }@`
 - このブロックは、AADLプログラム・ブロックの外側に記述します。
- JAVAアクションブロック：JAVAアクションブロックは、AADLの式の一部として記述できるブロック。
 - `@{ JAVAコード }@`
 - このブロックは、AADLの文法としては評価されないため、変数への代入では、変数の型にそのまま代入されるコードとなる。AADLでは、JAVAアクションブロックの内部は評価しないため、型情報が存在しません。そのため、内包記法のイテレータ式やエイリアスの記述には利用できない。

AADLエディタ使用方法

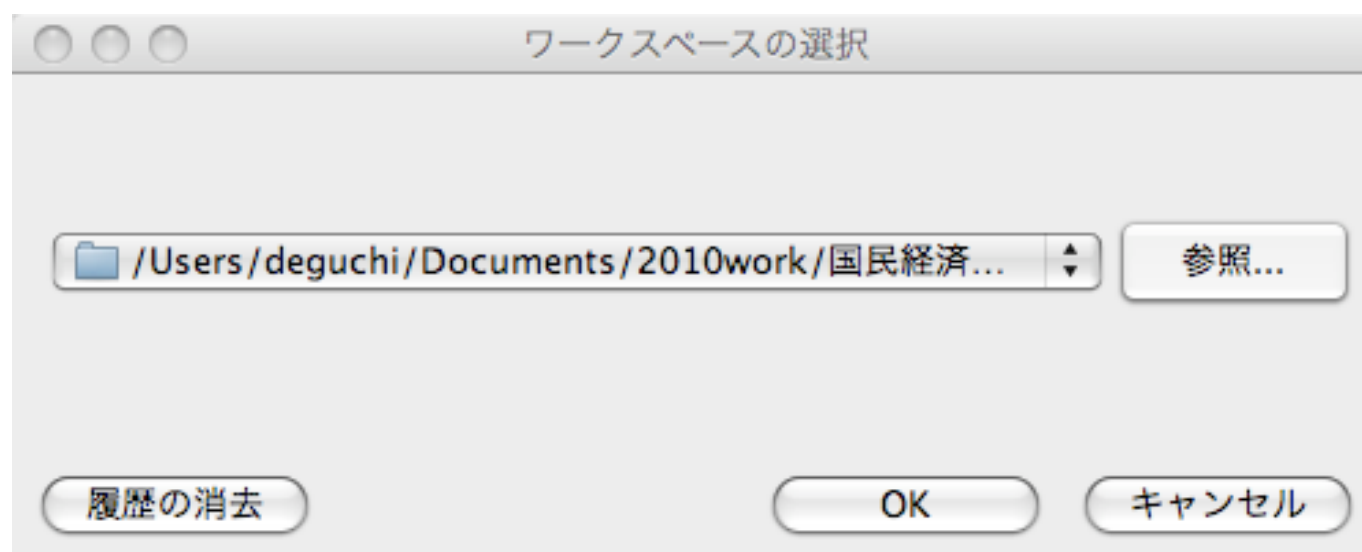
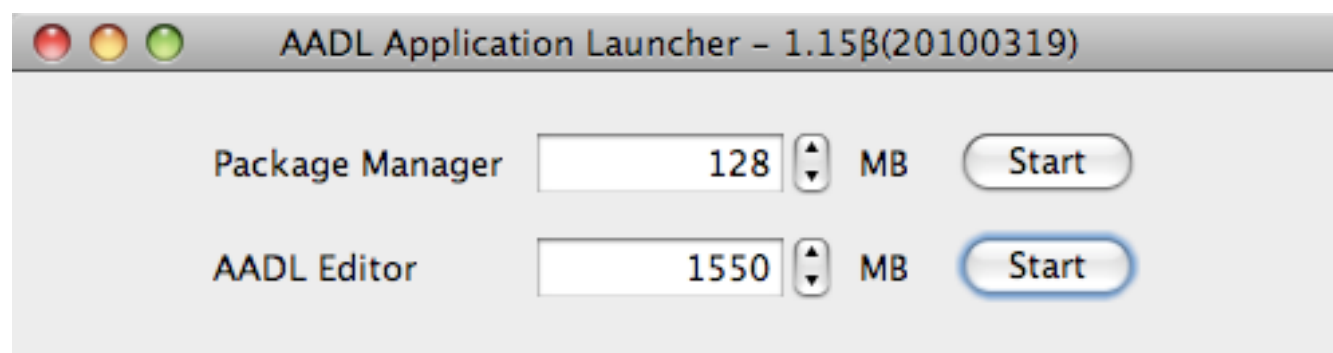
- AADLエディタは、複数のテキストをタブにより切り替え可能なエディタ。
- 複数のファイルを開くことができ、タブで編集対象を選択する。
- アクティブなタブのテキストが編集対象であり、コンパイルや実行の対象。
- AADLコンパイラは、本パッケージに含まれる。
- AADLエディタの実行は、本ファイルと同じフォルダにある"AADLEditor.jar"をダブルクリック、AADLエディタが起動する。

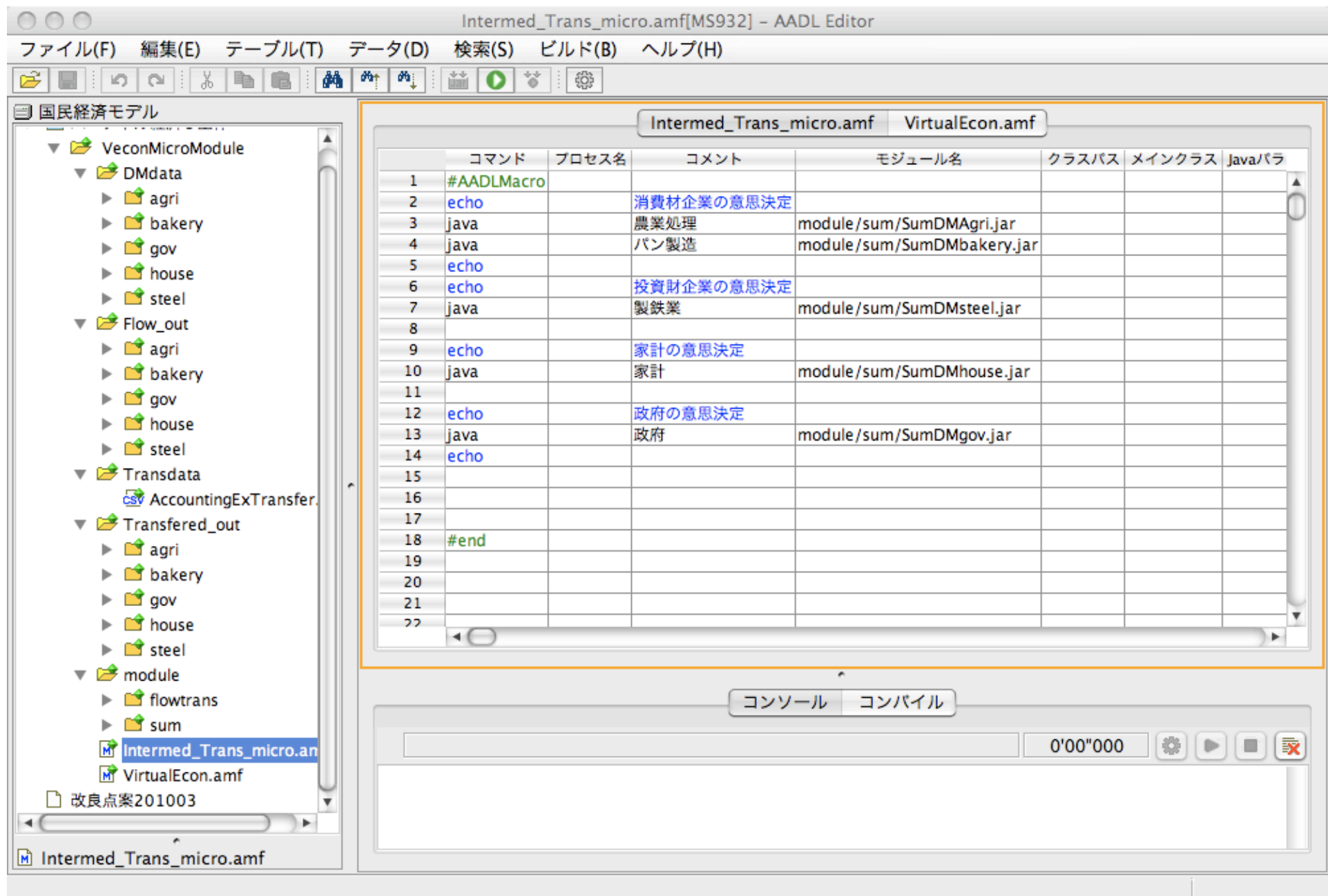
テキストの作成とコンパイル

- (1) 新規作成：メニュー[File]-[New]を選択すると新しいテキストが表示。
- (2) AADLソースコードを記述：新しいテキストにAADLソースコードを記述。よそのエディターで作った物のペーストも可能。
- (3) ファイルに保存：メニュー[File]-[SaveAs]を選択、テキストをファイルに保存。
- (4) コンパイル：メニュー[Build]-[Compile]を選択。新規テキストはファイルに保存されていない場合はコンパイルできない。
 - コンパイルに成功すると、AADLソースコードのファイルと同じ場所に、拡張子(*.jar)のファイルが生成される。
- <コンパイルメッセージ> コンパイルの結果は、画面下段の[Compile]タブの表示領域に表示される。この表示領域の右上の[×]ボタンをクリックすると、表示内容をクリアできる。
-

実行設定と実行

- 実行設定：メニュー[Build]-[Options]を選択、[Build options]ダイアログが表示され、[Run options]タブを選択。
- [Program arguments]にAADLの引数とするファイルを追加します。
- [+]ボタンでファイル選択ダイアログが表示されます。
- そのほかのボタンで、編集、削除、順序の入れ替えが可能です。
- 各ボタンの上にカーソルを合わせると、簡単な説明が表示されます。
- ※引数を必要としない場合は、追加する必要はありません。
- [Build options]ダイアログの[OK]ボタンをクリックすると、設定が保存され
- ダイアログが閉じます。
- (6) 実行
- メニュー[Build]-[Run]を選択します。
- 現在のAADL実行モジュールが実行されます。
- <実行メッセージ>





Aggregation Transfer by AADL Language

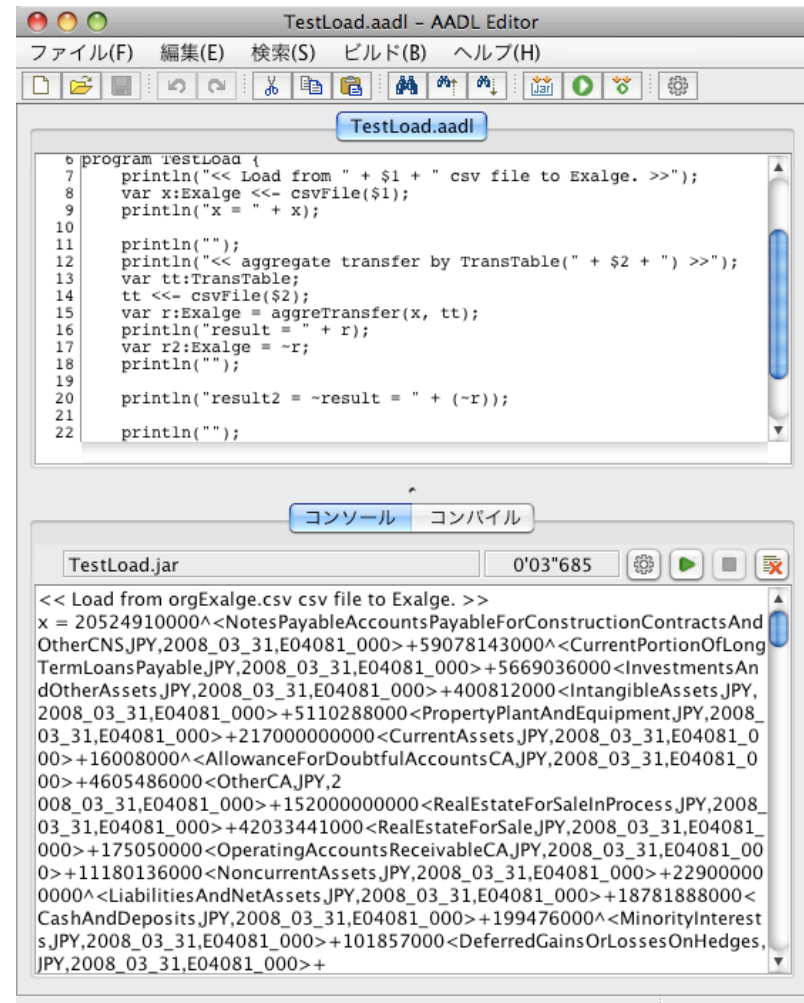
By using AADL we can easily calculate transfer operation

```
program TestLoad {  
  var x:Exalge <<- csvFile($1);  
  var tt:TransTable;  
  tt <<- csvFile($2);  
  var r:Exalge = aggreTransfer(x, tt);  
  var r2:Exalge = ~r;  
  r2 ->> csvFile($3);  
}
```

\$1:"orgExalge.csv"

\$2:"taxonomyTable.csv"

\$3:"resultExalge.csv"



The screenshot shows the AADL Editor interface. The top window, titled "TestLoad.aadl - AADL Editor", displays the source code of the "TestLoad" program. The code defines variables for "Exalge" and "TransTable", performs an aggregation transfer, and outputs the result to a CSV file. The bottom window, titled "コンソール" (Console), shows the execution output, including the load command and a large numerical result for the aggregation transfer operation.

```
TestLoad.aadl  
6 program TestLoad {  
7   println("<< Load from " + $1 + " csv file to Exalge. >>");  
8   var x:Exalge <<- csvFile($1);  
9   println("x = " + x);  
10  
11   println("");  
12   println("<< aggregate transfer by TransTable(" + $2 + ") >>");  
13   var tt:TransTable;  
14   tt <<- csvFile($2);  
15   var r:Exalge = aggreTransfer(x, tt);  
16   println("result = " + r);  
17   var r2:Exalge = ~r;  
18   println("");  
19  
20   println("result2 = ~result = " + (~r));  
21  
22   println("");  
}
```

TestLoad.jar 0'03"685

```
<< Load from orgExalge.csv csv file to Exalge. >>  
x = 20524910000^<NotesPayableAccountsPayableForConstructionContractsAnd  
OtherCNS,JPY,2008_03_31,E04081_000>+59078143000^<CurrentPortionOfLong  
TermLoansPayable,JPY,2008_03_31,E04081_000>+5669036000<InvestmentsAn  
dOtherAssets,JPY,2008_03_31,E04081_000>+400812000<IntangibleAssets,JPY,  
2008_03_31,E04081_000>+5110288000<PropertyPlantAndEquipment,JPY,2008_  
03_31,E04081_000>+217000000000<CurrentAssets,JPY,2008_03_31,E04081_0  
00>+16008000^<AllowanceForDoubtfulAccountsCA,JPY,2008_03_31,E04081_0  
00>+4605486000<OtherCA,JPY,2  
008_03_31,E04081_000>+152000000000<RealEstateForSaleInProcess,JPY,2008_  
03_31,E04081_000>+42033441000<RealEstateForSale,JPY,2008_03_31,E04081_  
000>+175050000<OperatingAccountsReceivableCA,JPY,2008_03_31,E04081_00  
0>+11180136000<NoncurrentAssets,JPY,2008_03_31,E04081_000>+22900000  
0000^<LiabilitiesAndNetAssets,JPY,2008_03_31,E04081_000>+18781888000<  
CashAndDeposits,JPY,2008_03_31,E04081_000>+1994760000^<MinorityInterest  
s,JPY,2008_03_31,E04081_000>+101857000<DeferredGainsOrLossesOnHedges,  
JPY,2008_03_31,E04081_000>+
```

Example of AADL Algebraic Specification of functional module

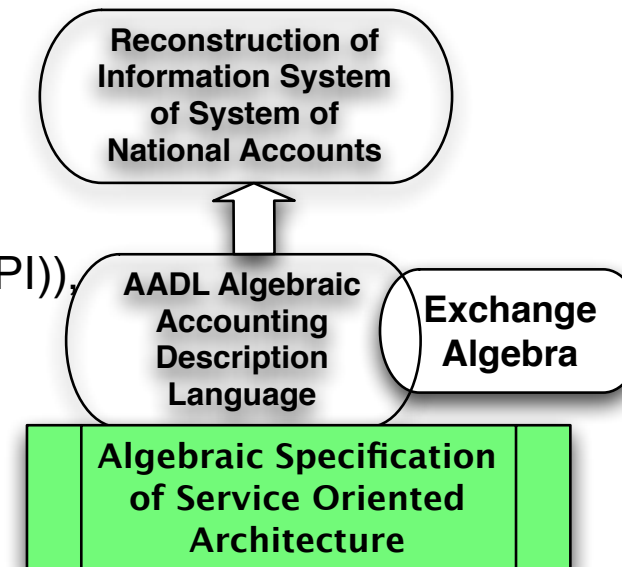
- Algebraic Specification of Statistical Calculation

$$Xq = \sum \left\{ q(e, t_0) < e, VQ, t_0, \# > \mid e \in \Lambda^{PI} \right\}$$

$$= \sum \left\{ \frac{\| \text{Pr}[< e, W, t_0, \# >](Xp) \|}{\| \text{Pr}[< e, PI, t_0, \# >](Xw) \|} < e, VQ, t_0, \# > \mid e \in \Lambda^{PI} \right\}$$

- Its AADL Description

```
var algeVQSet:ExAlgeSet
= { (na/nb)@<e, "VQ", refTime, "#"> | e<-basePI,
    na = norm(proj[<e,"WEIGHT",refTime, "#">](algePI)),
    nb = norm(proj[<e,"PI",refTime, "#">](algePI)),
    nb != 0 };
var algeVQ:Exalge = sum(algeVQSet);
```



事例：小さな国民経済の会計処理

- ここでは、農業、パン屋、製鉄、家計、政府からなる小さな国民経済の取引の会計処理のモデルをシミュレートしてみよう。
- エージェントとして、農業、パン屋、製鉄、家計、政府を想定し、国民経済全体を表すスポットを導入し、そこに適当な交換代数の変数を導入し、この国民経済を記述する最小限の取引を導入する。

期首の状態

農業	期首貸借対照表		
借方	金額	貸方	金額
現金	50	資本金	100
機械（鉄）	50		
小計	100		100

家計	期首貸借対照表		
借方	金額	貸方	金額
現金	100	資本金	130
機械（鉄）	30		
小計	130		130

パン屋	期首貸借対照表		
借方	金額	貸方	金額
現金	50	資本金	100
機械（鉄）	50		
小計	100		100

製鉄	期首貸借対照表		
借方	金額	貸方	金額
現金	50	資本金	100
機械（鉄）	50		
小計	100		100

政府	期首貸借対照表		
借方	金額	貸方	金額
現金	50	資本金	70
機械（鉄）	20		
小計	70		70

簿記表現と交換代数との対応

期首貸借対照表 1

農業	期首貸借対照表		
借方	金額	貸方	金額
現金	50	資本金	100
機械（鉄）	50		
小計	100		100

$bl_beginoftem_agri$
 $=50<現金,yen,\#,agri>$
 $+100<資本金,yen,\#,agri>$
 $+50<機械,yen,\#,agri>$

パン屋	期首貸借対照表		
借方	金額	貸方	金額
現金	50	資本金	100
機械（鉄）	50		
小計	100		100

$bl_beginofterm_bakery$
 $=50<現金,yen,\#,bakery>$
 $+100<資本金,yen,\#,bakery>$
 $+50<機械,yen,\#,bakery>$

製鉄	期首貸借対照表		
借方	金額	貸方	金額
現金	50	資本金	100
機械（鉄）	50		
小計	100		100

$bl_beginofterm_steel$
 $=50<現金,yen,\#,steel>$
 $+100<資本金,yen,\#,steel>$
 $+50<機械,yen,\#,steel>$

簿記表現と交換代数との対応

期首貸借対照表 2

家計	期首貸借対照表		
借方	金額	貸方	金額
現金	100	資本金	130
機械（鉄）	30		
小計	130		130

$bl_beginofterm_house$
 $=100<\text{現金}, yen, \#, house>$
 $+130<\text{資本金}, yen, \#, house>$
 $+30<\text{機械}, yen, \#, house>$

政府	期首貸借対照表		
借方	金額	貸方	金額
現金	50	資本金	70
機械（鉄）	20		
小計	70		70

$bl_beginofterm_gov$
 $=50<\text{現金}, yen, \#, gov>$
 $+70<\text{資本金}, yen, \#, gov>$
 $+20<\text{機械}, yen, \#, gov>$

簿記表現と 交換代数との対応

	農業	期中取引		
取引分類	借方	金額	貸方	金額
Production	小麦	80	付加価値	80
SellProduct	現金	70	小麦	70
Wage	労賃支出	50	現金	50
Investment	鉄	20	現金	20
Tax	税支出	5	現金	5

パン屋	期中取引		
借方	金額	貸方	金額
小麦	70	現金	70
パン	140	付加価値	70
		小麦	70
現金	140	パン	140
労賃支出	40	現金	40
鉄	20	現金	20
税支出	5	現金	5

$\text{production_agri} = 80 \langle \text{小麦, yen, \#, agri} \rangle$
 $+ 80 \langle \text{付加価値, yen, \#, agri} \rangle$
 $\text{sellproduct_agri} = 70^{\wedge} \langle \text{小麦, yen, \#, agri} \rangle$
 $+ 70 \langle \text{現金, yen, \#, agri} \rangle$
 $\text{wage_agri} = 50 \langle \text{労賃支出, yen, \#, agri} \rangle$
 $+ 50^{\wedge} \langle \text{現金, yen, \#, agri} \rangle$
 $\text{invest_agri} = 20 \langle \text{機械, yen, \#, agri} \rangle$
 $+ 20^{\wedge} \langle \text{現金, yen, \#, agri} \rangle$
 $\text{tax_agri} = 5 \langle \text{税支出, yen, \#, agri} \rangle$
 $+ 5^{\wedge} \langle \text{現金, yen, \#, agri} \rangle$

注：本来の位置にある勘定科目（つまりその勘定の増大）はNO_HAT
 が、反対側にある場合（その勘定の減少）にはHATがつく。

振替処理の補完

振替処理	取引分類	借方	金額	貸方	金額
農業	TransValue	付加価値	80	内部留保	80
農業	TransWage	内部留保	50	労賃支出	50
農業	TransTax	内部留保	5	税支出	5
パン屋	TransValue	付加価値	70	内部留保	70
パン屋	TransWage	内部留保	40	労賃支出	40
パン屋	TransTax	内部留保	5	税支出	5
製鉄	TransValue	付加価値	70	内部留保	70
製鉄	TransWage	内部留保	45	労賃支出	45
製鉄	TransTax	内部留保	5	税支出	5
家計	TransWage	労賃収入	150	内部留保	150
家計	TransConsump	内部留保	140	最終消費	140
家計	TransTax	内部留保	5	税支出	5
政府	TransValue	付加価値	15	内部留保	15
政府	TransTax	税収入	20	内部留保	20
政府	TransWage	内部留保	15	労賃支出	15
政府	TransConsump	内部留保	15	最終消費	15

収入、支出の内部留保への振替処理を行う。この取引も、機械的に行える。

簿記表現期中取引 2

政府	期中取引			
借方	金額	貸方	金額	
現金	5	税収入	5	製鉄から
現金	5	税収入	5	農業から
現金	5	税収入	5	パン屋から
現金	5	税収入	5	家計から
労賃支出	15	現金	15	
政府サービス	15	付加価値	15	
最終消費	15	政府サービス	15	
鉄	10	現金	10	

家計	期中取引			
借方	金額	貸方	金額	
現金	50	労賃収入	50	農業労賃
現金	40	労賃収入	40	パン屋労賃
現金	45	労賃収入	45	製鉄労賃
現金	15	労賃収入	15	政府労賃
パン	140	現金	140	
最終消費	140	パン	140	
鉄	5	現金	5	
税支出	5	現金	5	

製鉄	期中取引			
借方	金額	貸方	金額	
鉄	70	付加価値	70	
現金	15	鉄	15	製鉄へ
現金	20	鉄	20	農業へ
現金	20	鉄	20	パン屋へ
現金	5	鉄	5	家計へ
現金	10	鉄	10	政府へ
労賃支出	45	現金	45	
鉄	15	現金	15	
税支出	5	現金	5	

振替処理

	農業	振替処理		
取引分類	借方	金額	貸方	金額
TransValue	付加価値	80	内部留保	80
TransWage	内部留保	50	労賃支出	50
TransTax	内部留保	5	税支出	5

パン屋	振替処理		
借方	金額	貸方	金額
付加価値	70	内部留保	70
内部留保	40	労賃支出	40
内部留保	5	税支出	5

製鉄	振替処理		
付加価値	70	内部留保	70
内部留保	45	労賃支出	45
内部留保	5	税支出	5

家計	振替処理		
労賃収入	150	内部留保	150
内部留保	140	最終消費	140
内部留保	5	税支出	5

政府	振替処理		
付加価値	15	内部留保	15
税収入	20	内部留保	20
内部留保	15	労賃支出	15
内部留保	15	最終消費	15

交換代数との対応（振替）

付加価値振替

$\text{transvalue_agri} = 80 \langle \text{内部留保, yen, \#, agri} \rangle + 80 \wedge \langle \text{付加価値, yen, \#, agri} \rangle$

労賃支出振替

$\text{transwage_agri} = 50 \wedge \langle \text{労賃支出, yen, \#, agri} \rangle + 50 \wedge \langle \text{内部留保, yen, \#, agri} \rangle$

	農業	振替処理		
取引分類	借方	金額	貸方	金額
TransValue	付加価値	80	内部留保	80
TransWage	内部留保	50	労賃支出	50
TransTax	内部留保	5	税支出	5

税支出振替

$\text{transtax_agri} = 5 \wedge \langle \text{税支出, yen, \#, agri} \rangle + 5 \wedge \langle \text{内部留保, yen, \#, agri} \rangle$

期中フロー計算

農業	期中フロー：残高試算表		
借方	金額	貸方	金額
小麦	10	内部留保	25
現金	0	小麦	0
鉄	20	現金	5
小計	30		30

家計	期中フロー：残高試算表		
借方	金額	貸方	金額
現金	0	現金	0
鉄	5	内部留保	5
小計	5	小計	5

パン屋	期中フロー：残高試算表		
借方	金額	貸方	金額
パン	0	内部留保	25
現金	5	小麦	0
鉄	20		
小計	25		25

製鉄	期中フロー：残高試算表		
借方	金額	貸方	金額
鉄		鉄	0
現金	5	現金	0
鉄	15	内部留保	20
小計	20	小計	20

政府	期中フロー：残高試算表		
借方	金額	貸方	金額
現金		現金	5
鉄	10	内部留保	5
小計	10	小計	10

期末貸借対照表

農業	期末貸借対照表		
借方	金額	貸方	金額
借方	金額	貸方	金額
現金	45	資本金	100
機械（鉄）	70	内部留保	25
小麦（在庫）	10		
小計	125		125

家計	期末貸借対照表		
借方	金額	貸方	金額
借方	金額	貸方	金額
現金	100	資本金	130
機械（鉄）	35	内部留保	5
小計	135		135

パン屋	期末貸借対照表		
借方	金額	貸方	金額
借方	金額	貸方	金額
現金	55	資本金	100
機械(鉄)	70	内部留保	25
小麦(在庫)	0		
小計	125		125

製鉄	期末貸借対照表		
借方	金額	貸方	金額
借方	金額	貸方	金額
現金	55	資本金	100
機械（鉄）	65	内部留保	20
小計	120		120

政府	期末貸借対照表		
借方	金額	貸方	金額
借方	金額	貸方	金額
現金	45	資本金	70
機械（鉄）	30	内部留保	5
小計	75		75

国民経済計算 $Y=I+C=S+C, I=S$

期首国民貸借対照表			
借方	金額	貸方	金額
現金	300		
小麦	0	資本金	500
鉄（設備）	200		

国民経済	期中フロー：残高試算表		
借方	金額	貸方	金額
パン	0	内部留保	80
小麦	10	小麦	0
鉄	70	現金	0

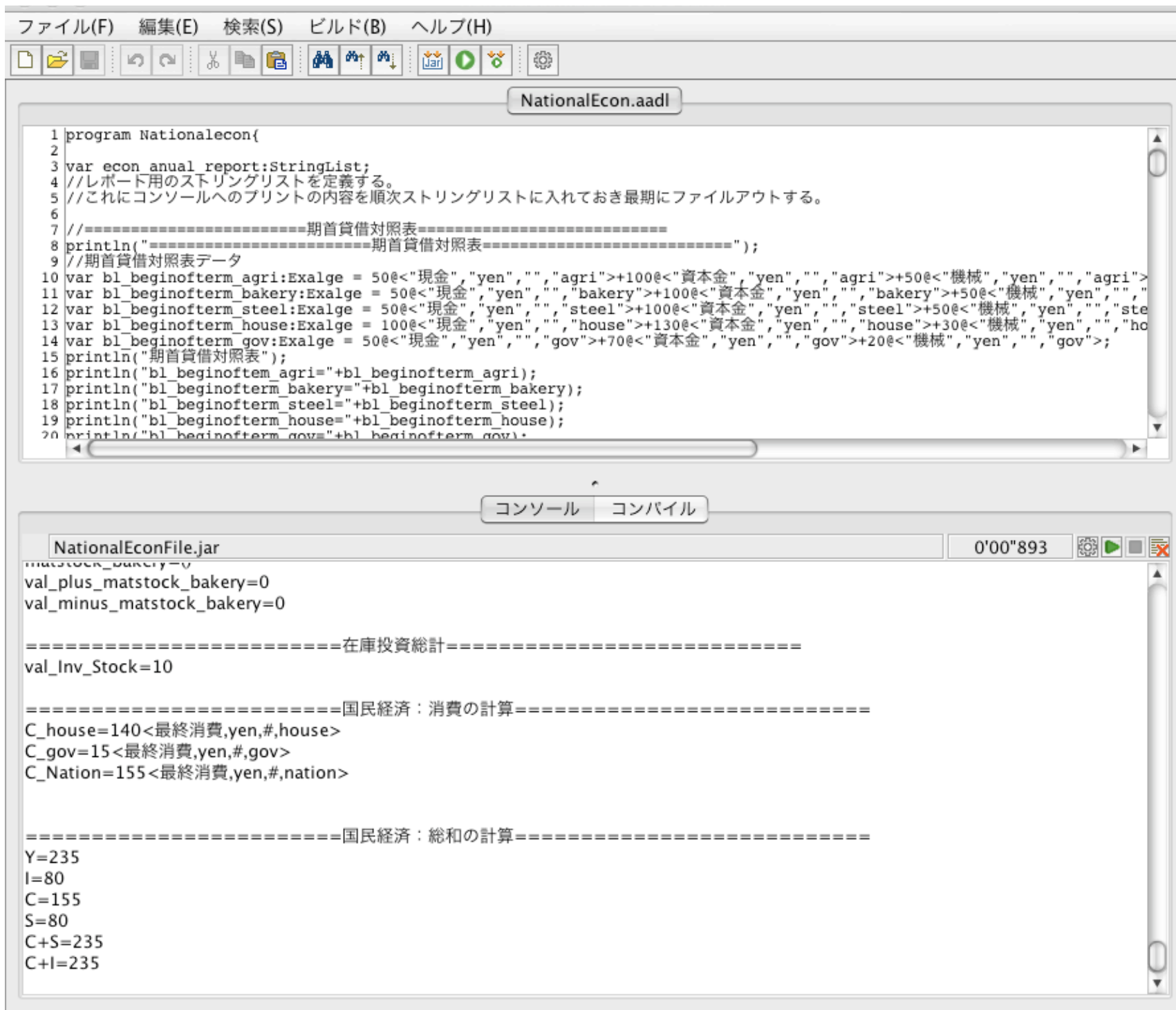
国民経済	期末貸借対照表		
借方	金額	貸方	金額
現金	300	資本金	500
機械（鉄）	270	内部留保	80
小麦（在庫）	10		

GTP		
Y[Steel]	70	付加価値
Y[Agri]	80	付加価値
Y[Bakery]	70	付加価値
Y[Gov]	15	付加価値
Y[SNA]	235	

貯蓄（内部留保）		
S[Steel]	20	内部留保
S[Agri]	25	内部留保
S[Bakery]	25	内部留保
S[House]	5	内部留保
S[Gov]	5	内部留保
S[SNA]	80	

設備投資		
I[Steel]	15	製鉄へ
I[Agri]	20	農業へ
I[Bakery]	20	パン屋へ
I[House]	5	家計へ
I[Gov]	10	政府へ
在庫投資		
I[pro;Steel]	0	鉄製品在庫
I[pro;Agri]	10	小麦製品在庫
I[pro;Bakery]	0	パン製品在庫
I[mat;Bakery]	0	パン原料在庫
I[SNA]	80	

消費		
C[Gov]	15	最終消費
C[House]	140	最終消費
C[SNA]	155	



産業連関表による経済構造

第1表 取引基本表
(単位:億円)

		中間需要		最終需要	生産額
		A産業	B産業		
中間投入	A産業	30	150	120	300
	B産業	60	250	190	500
粗付加価値		210	100		
生産額		300	500		

		需要 (買い手)		最終需要	地域内生産額
		産業1	産業2		
中間投入	産業1	x_{11}	x_{12}	F_1	X_1
	産業2	x_{21}	x_{22}	F_2	X_2
粗付加価値		V_1	V_2		
地域内生産額		X_1	X_2		

[http://www.stat.go.jp/data/
io/system.htm](http://www.stat.go.jp/data/io/system.htm)

経済システムに於けるマイクロマクロ リンクと動学の新たな研究プログラム

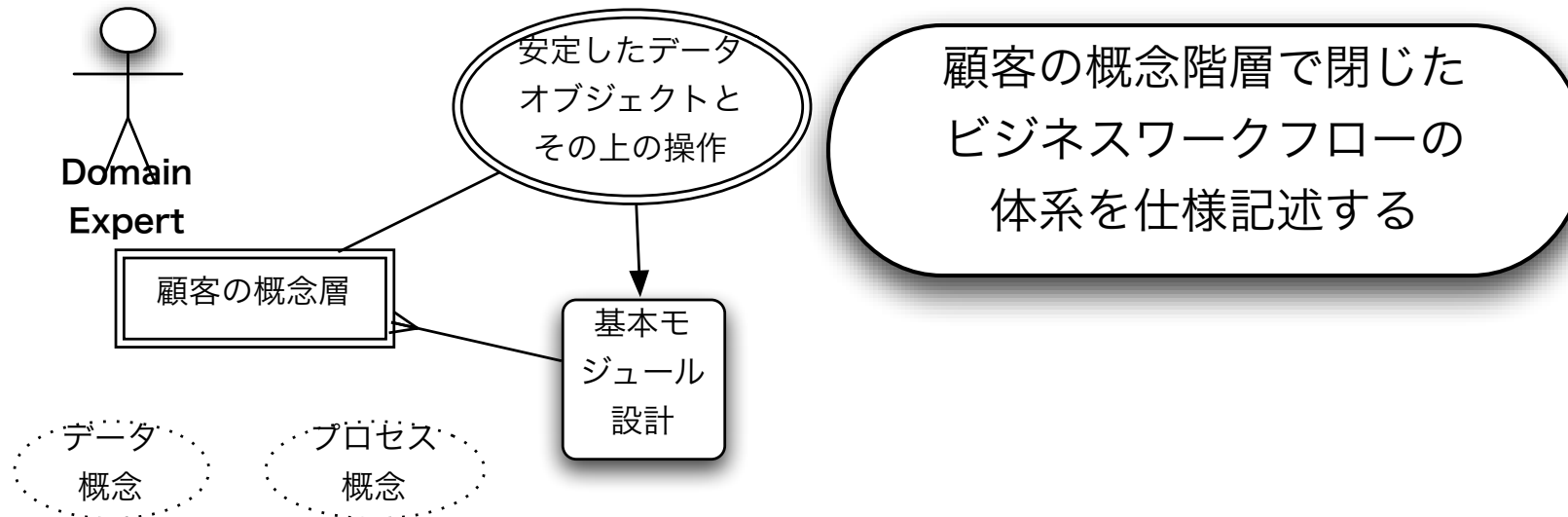
経済状態のトランザクション記述

- 経済主体の状態表現とその変化を記述する基礎的な枠組みに簿記によるストック状態の複式記述と、経済主体間の取引としてのフローの記述がある。経済システムは、各経済主体のマイクロなストック状態と、経済主体の間の様々な財の生産、流通、消費に関する取引の連鎖として補足されるフローによりマイクロには記述される。
- このような取引(Transaction)ベースでの経済システムの状態記述は、R. マテシッチが構想した会計経済学の枠組みなど歴史上幾つか散見するが、今日の経済システム記述の主流とはならなかった。
- 他方で経済システムの加工統計的な記述としての国民経済計算(SNA)は、工業統計表などの経済主体に対する、基幹統計調査（旧指定統計、今後の経済センサス）の個票に基づき、付加価値や産出などに関するボトムアップな経済のフローストックの状態推計を行う。ここでは、ボトムアップな経済状態の推計が社会会計的な視点から行われるが、そこでの原データは、経済活動の主体間の簿記的な意味での取引記述では必ずしもないし推計の論理もアドホックな部分も多い。

交換代数による状態記述

- これに対して我々は交換代数という簿記を抽象化した代数体系に基づいた経済主体の状態記述とフロー記述からボトムアップに経済システムのマクロ状態記述を構成するという視点から経済システムの状態記述を行う枠組みを提起している。
- ただしこれは状態記述のレベルであり、マイクロなビジネスコンポーネント単位での状態記述からスタートして、そこから組織内、組織間のビジネスコンポーネントの連結構造やその動的なシステム把握や、そこでのビジネスコンポーネントの進化や、構造変化、さらに制度設計を論じるにはまだ長いステップが必要となる。
- ここでの立場はまず経済の状態記述を、マイクロな会計的な状態記述から首尾一貫して、マクロな社会会計まで再構築することである。それにより様々な経済的相互作用を交換の連鎖の回路として分析し、かつデザインする枠組みを構築することにある。

サービス指向の階層アーキテクチャ で様々なシナリオ解析モデルを開拓する



- 社会統計の記述の為に簿記の抽象化（交換代数）ベースの代数的仕様記述を採用：常に変化し、更に情報爆発しつつある社会会計の大規模業務システムを設計
- 社会統計データを基盤とすることでエージェントベースの大規模経済モデルがリアルなものとなる

データさえ
あれば！

社会シミュレーションが社会の
基幹インフラとなる時代が必ず
やってくる

政策や意思決定のために
に活用するためのシ
ナリオオプションの予
測と評価モデルの層

シナリオオプション
の予測評価モデル層

フィード
フォワード

政策シナリ
オの予測評
価モデル

経済予測モデル

医療費と医療保
険の予測モデル

交通量予測

年金保険予測

人口動態予測

政策ニーズに応じて予
測評価モデルは様々

フィードバック 利用

相互主観的なデータとそ
の関連を構成的にモデル
化したデータモデル層

交換代数とAADL

データ
モデル層

二次統計：構
成的なデー
タモデル

二次統計として
構成される基幹
統計モデル

エージェントベースシミュレーション利用

社会人口
統計体系

政策目的に応じて基幹統計
のデータモデルは異なる

国民経済

社会経済の諸現象から
データを測定する層

データ代数とADDL

データ
測定層

一次統計：基幹統計
調査データやPOSな
どのスキャナデー
タ、業務統計データ

一次データの測定はデータモ
デルのあり方に依存する
Model Ladenness

社会シミュレーションによるシナ
リオ推計が社会の標準的な政策立
案と討議、合意形成ツールとしてイ
ンプリメントされる社会へ向けて

状態推計が
シナリオ
推計へ推計
の高度化
シナリオ
推計層

データ
モデル層

データ
測定層

既存の制度統計体系

基幹
統計
基幹
統計
調査

加工統計

従来の調査票に
よる統計調査

電子スキャニング
統計の確立とその
上のデータの高度
加工のシステム化

短い時定数
での推計へ

Convert to
Exchange
Algebra

Crawling

Agent Based
Simulation

Wall Street
XBRL data of
companies
EDINET in
Japan

EDINET (Electronic
Disclosure for
Investors' NETwork)

高次将来推計としての
社会シミュレーション

SNA等の
二次統計

一次統計
の調査

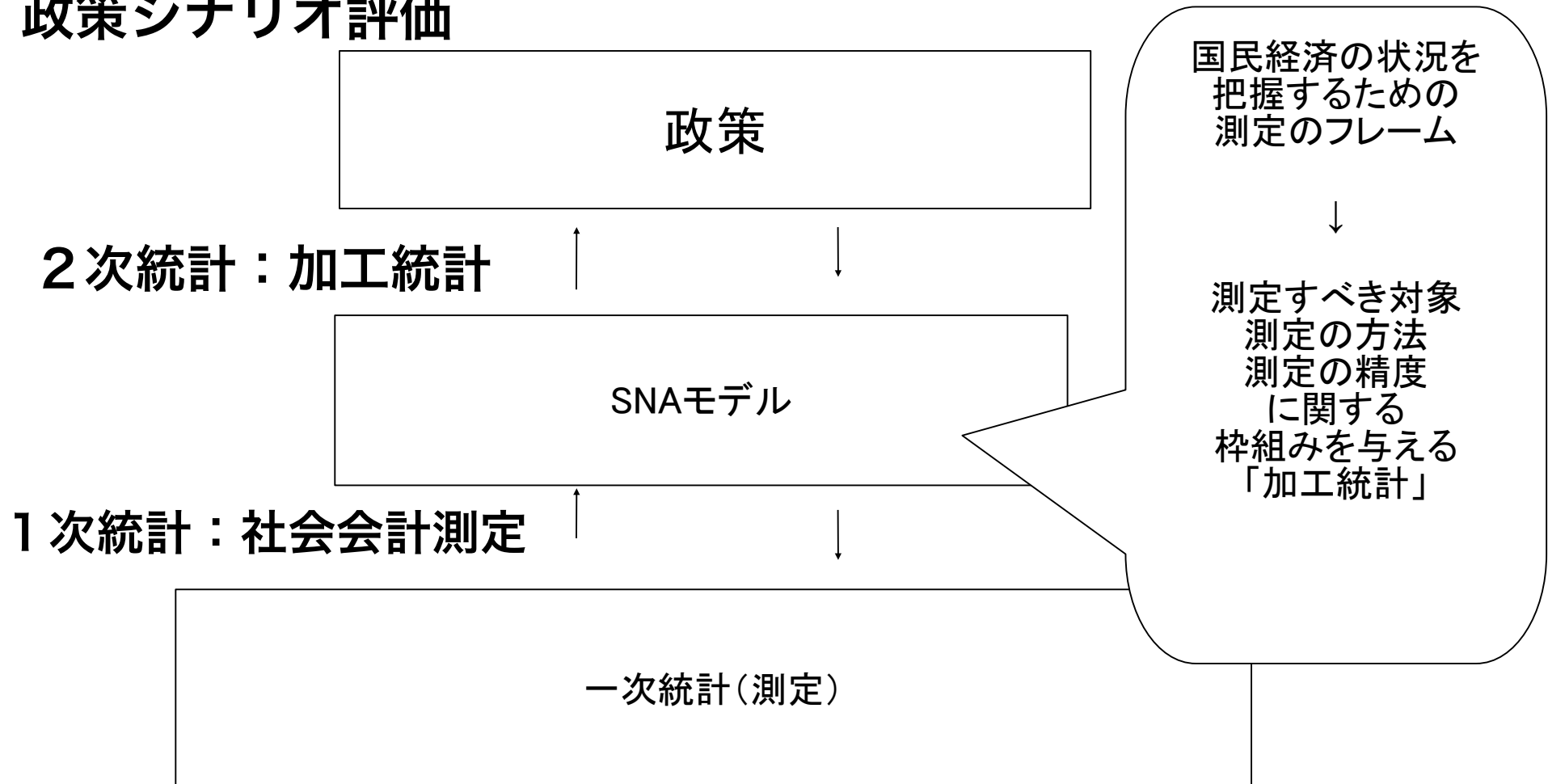
データモデル
層の高度加工
AADL

データ測定層
のクラウド
ソーシング

会計的マイクロデータ
からの変換抽出

3層の統計利活用の事例としての 国民経済計算

政策シナリオ評価



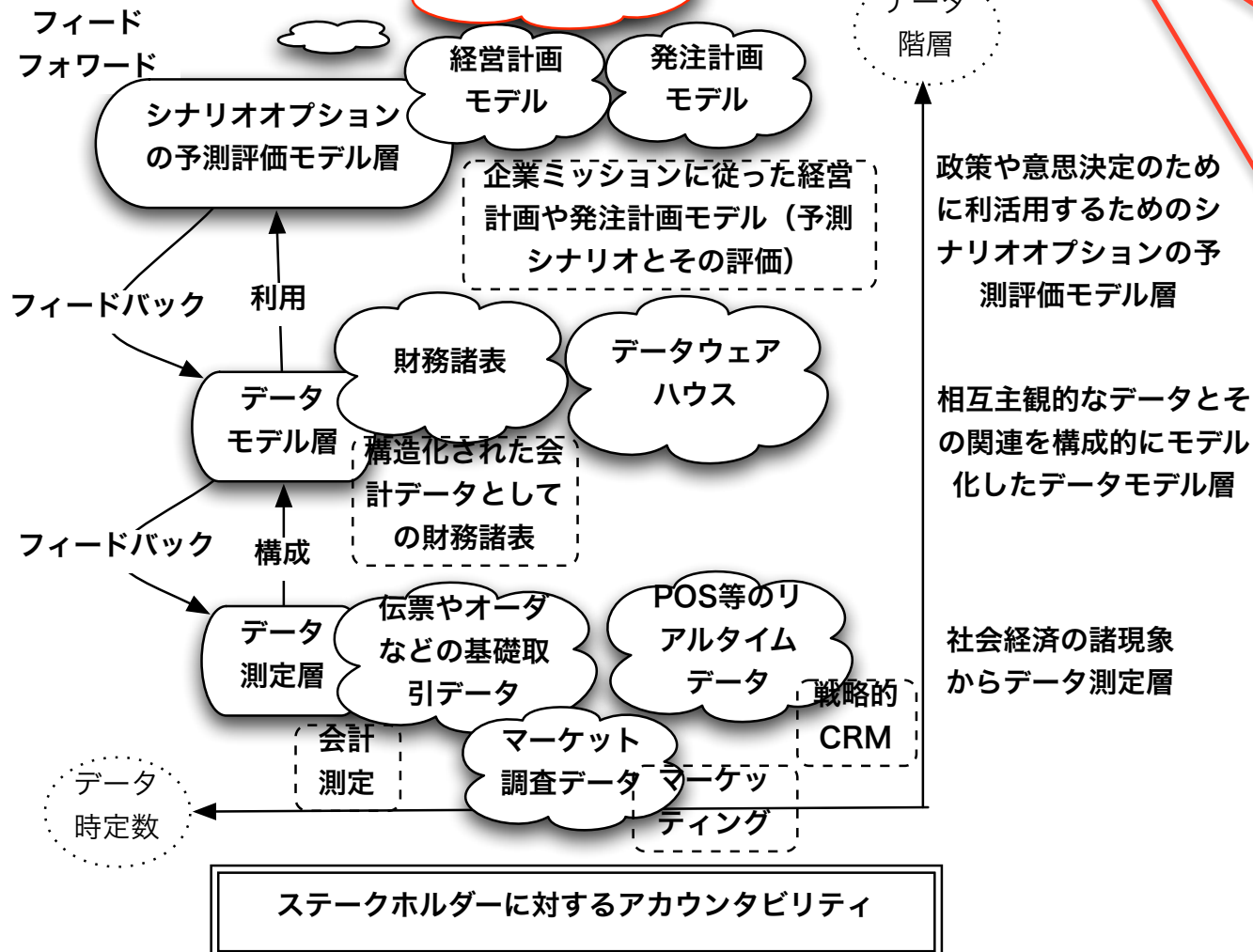
企業の戦略意思決定のための統計 情報システムアーキテクチャ

As Is

事業所別データの
情報システム
によ統合は基本

時定数の異なるデータの収
集と利用が行動計画、投資
計画、危機管理等に必須

紙ベースの調査やデータ記
録収集から、電子化された
データ利用への移行



情報爆発時代の国民経済計算システムの設計：

モデル化のための仕様設計と実装分析

ー内閣府経済社会総合研究所との連携で

社会統計データ処理システム設計ー

www.esri.cao.go.jp/jp/sna/070524/hossoku.pdf

平成19年5月24日. 内閣府経済社会総合研究所. 1. 概要. 内閣府経済社会総合研究所は平成19年5月29日、東工大エージェントベース社会システム科学研究センターとの共同で社会会計システム・オープン・コ.ンソーシアムを設置。

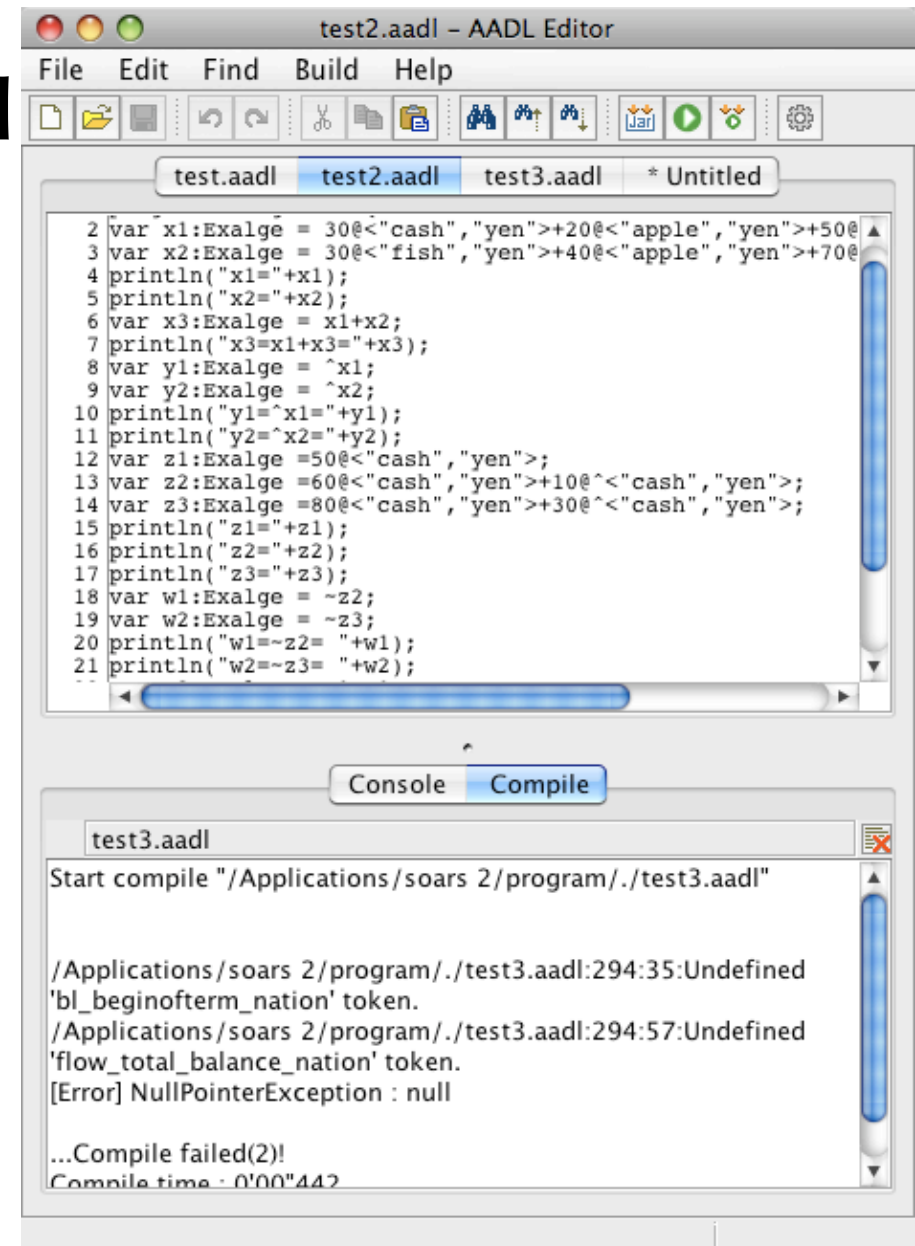
- ✻ 現在内閣府の国民経済計算(SNA)の情報システムの再構築のデザインに参与している。

状態記述のアプローチ 2

- 3) 我々の経済モデルは、経済主体の意思決定については、「規範的」「合理的」な方法論的個人主義の立場を取らない。かといって決まったモデル枠組みでのパラメータ推定でもない。システムの分析には、状態記述を厳密にした上でのシミュレーションという方法を積極的に用いる。経済主体の意思決定はそこでは固定的な枠組みやパラメータとしてではなく、陽にモデルに組込むことでその変化そのものも分析の対象とする。
- 4) 実際のモデル構築では、数理言語だけでなくシミュレーション言語によるモデルの記述が、ボトムアップなアプローチでは必須となる。そのためにまず状態記述の数理（交換代数）をジャバのパッケージ化した交換代数と、その上の会計状態の記述言語である、AADL(Algebraic Accounting Description Language:代数的会計記述言語) というプログラム言語を既に我々は開発している。これを用いることで個々の主体の会計レベルの記述から、国民経済の推計モデルの再構築まで幅広い用途で、交換代数に基づく経済主体の状態や取引の代数的な記述を容易にモデル化できる。

AADL as DSL (Domain Specific Language)

- Not for Interface Oriented DSL(Domain Specific Design Language)
- But for Specification Description Language depending of Data Structure for Domain Expert, AADL(Algebraic Accounting Description/ Design Language) has developed.

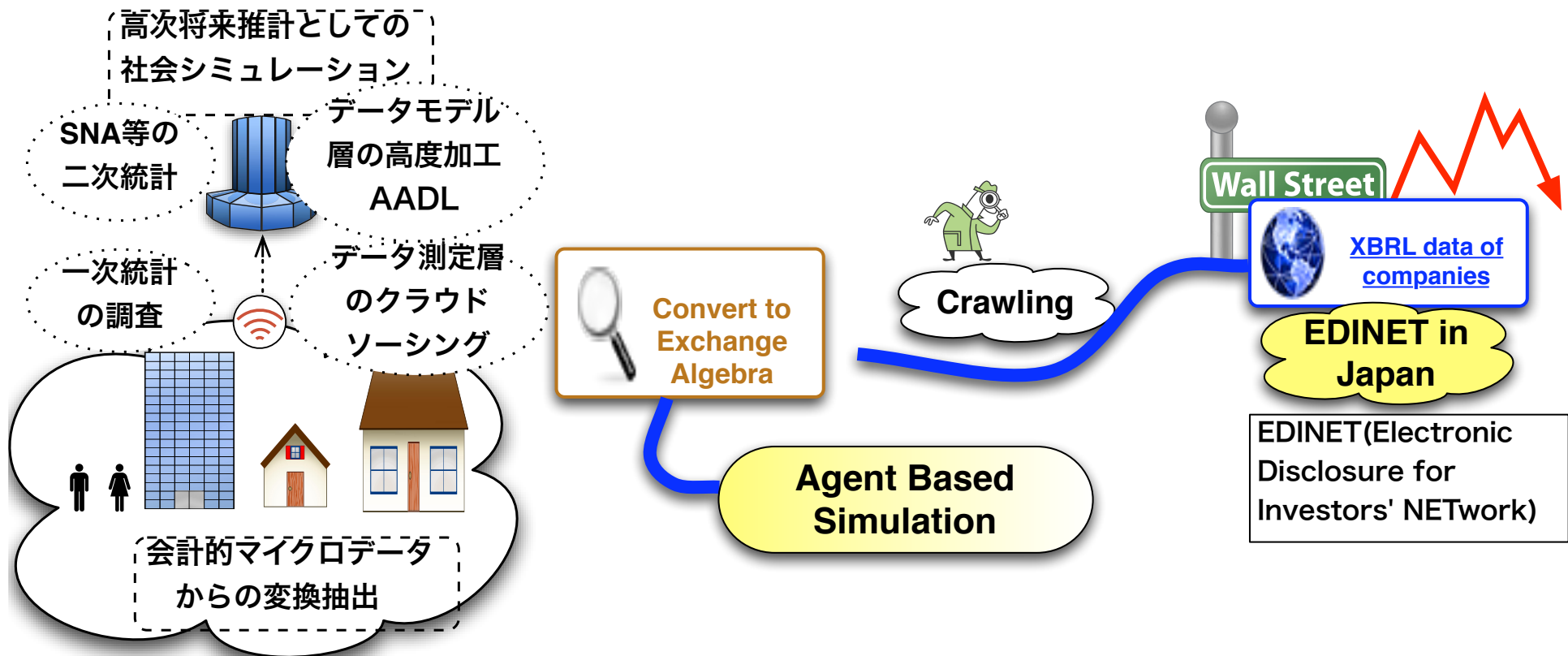


実際の会社の会計状態の補足

- 企業を一つのビジネスコンポーネントと見たレベルでの会計状態については、金融庁のEDINET（Electronic Disclosure for Investors' NETwork）上に上場企業の四半期ベースでの財務諸表がXBRLにより開示されている。
- XBRLは表章には使えても、その形式で複雑な計算をすることには向かない。何よりも交換記述を表現する数理的な構造に依拠していない。我々はこのデータを交換代数に変換して、AADL上で簡単に操作でき、上場企業の会計状態の串刺しの計算等を容易に行うことができる。
- また米国でも企業の会計状態のXBRLによる開示は進み始めており、これらとエージェントベースのシミュレーションモデルを組み合わせる事で、従来は不可能だったボトムアップの経済モデルの構築も視野に入ってきた。

XBRL:Crawling & Convert to AADL

- XBRL is easy to express and exchange accounting data. On the other hand it is difficult to calculate the data.
- AADL language is designed on exchange algebra for handling social and organizational accounting data.



AADL言語でのアグリゲーション

Aggregation Transfer by AADL Language

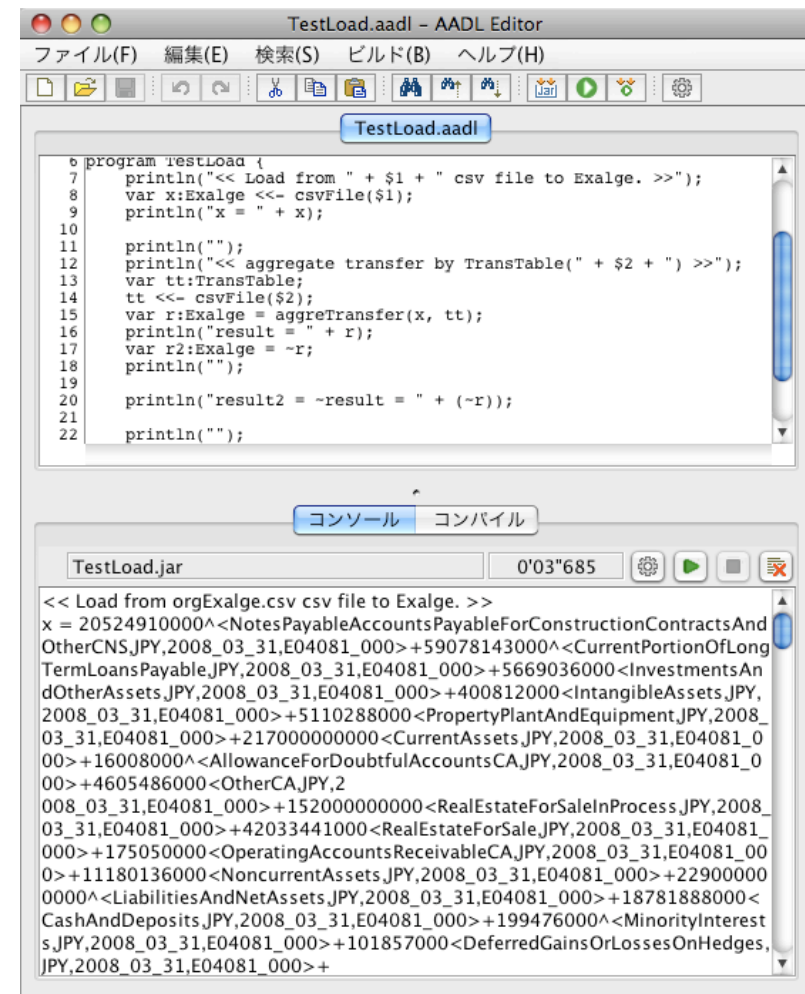
交換代数CSV標準形と振替変換テーブルCSV標準形を読み込み、
振替変換を実施した結果を保存する、単純なAADL。

```
program TestLoad {  
  var x:Exalge <<- csvFile($1);  
  var tt:TransTable;  
  tt <<- csvFile($2);  
  var r:Exalge = aggreTransfer(x, tt);  
  var r2:Exalge = ~r;  
  r2 ->> csvFile($3);  
}
```

\$1:"orgExalge.csv"

\$2:"taxonomyTable.csv"

\$3:"resultExalge.csv"



The screenshot shows the AADL Editor interface. The top window, titled "TestLoad.aadl - AADL Editor", displays the source code of the "TestLoad" program. The code defines variables for "Exalge" and "TransTable", loads data from CSV files, performs an aggregation transfer, and saves the result. The bottom window, titled "コンソール" (Console), shows the output of the program execution, including the load status and the resulting "Exalge" data structure.

```
TestLoad.aadl  
6 program testLoad {  
7   println("<< Load from " + $1 + " csv file to Exalge. >>");  
8   var x:Exalge <<- csvFile($1);  
9   println("x = " + x);  
10  
11   println("");  
12   println("<< aggregate transfer by TransTable(" + $2 + ") >>");  
13   var tt:TransTable;  
14   tt <<- csvFile($2);  
15   var r:Exalge = aggreTransfer(x, tt);  
16   println("result = " + r);  
17   var r2:Exalge = ~r;  
18   println("");  
19  
20   println("result2 = ~result = " + (~r));  
21  
22   println("");  
}
```

TestLoad.jar 0'03"685

```
<< Load from orgExalge.csv csv file to Exalge. >>  
x = 20524910000^<NotesPayableAccountsPayableForConstructionContractsAnd  
OtherCNSJPY,2008_03_31,E04081_000>+59078143000^<CurrentPortionOfLong  
TermLoansPayableJPY,2008_03_31,E04081_000>+5669036000<InvestmentsAn  
dOtherAssetsJPY,2008_03_31,E04081_000>+400812000<IntangibleAssetsJPY,  
2008_03_31,E04081_000>+5110288000<PropertyPlantAndEquipmentJPY,2008  
_03_31,E04081_000>+217000000000<CurrentAssetsJPY,2008_03_31,E04081_0  
00>+16008000^<AllowanceForDoubtfulAccountsCAJPY,2008_03_31,E04081_0  
00>+4605486000<OtherCAJPY,2  
008_03_31,E04081_000>+152000000000<RealEstateForSaleInProcessJPY,2008  
_03_31,E04081_000>+42033441000<RealEstateForSaleJPY,2008_03_31,E04081  
_000>+175050000<OperatingAccountsReceivableCAJPY,2008_03_31,E04081_00  
0>+11180136000<NoncurrentAssetsJPY,2008_03_31,E04081_000>+22900000  
0000^<LiabilitiesAndNetAssetsJPY,2008_03_31,E04081_000>+18781888000<  
CashAndDepositsJPY,2008_03_31,E04081_000>+199476000^<MinorityInterest  
sJPY,2008_03_31,E04081_000>+101857000<DeferredGainsOrLossesOnHedges,  
JPY,2008_03_31,E04081_000>+
```

タクソミーを用いたアグリゲーション型の振替

Aggregation by using AADL depending on converted taxonomy data

"jpfr-q2r-E04081-000-2008-09-30-01-2009-01-23_taxonomy.csv" の
内容を、TransTableのCSV標準形に整形したもの。

from	Assets
to	AssetsAbstract
from	LiabilitiesAndNetAssets
to	BalanceSheetsAbstract
from	CurrentAssets
from	AllowanceForDoubtfulAccountsCA
from	OtherCA
from	RealEstateForSaleInProgress
from	RealEstateForSale
from	OperatingAccountsReceivableCA
from	CashAndDeposits
to	CurrentAssetsAbstract
from	NotesPayableAccountsPayableForConstructionContractsAndOtherCNS
from	CurrentPortionOfLongTermLoansPayable
from	OtherCL
from	CurrentLiabilities
from	ProvisionCL
from	IncomeTaxesPayable
from	ShortTermLoansPayable
to	CurrentLiabilitiesAbstract
from	IntangibleAssets
to	IntangibleAssetsAbstract
from	Liabilities
to	LiabilitiesAbstract
from	MinorityInterests
from	NetAssets
to	NetAssetsAbstract
from	InvestmentsAndOtherAssets
#from	IntangibleAssets
from	PropertyPlantAndEquipment
from	NoncurrentAssets
to	NoncurrentAssetsAbstract
from	NoncurrentLiabilities
from	OtherNCL
from	ProvisionNCL
from	LongTermLoansPayable
from	BondsPayable
to	NoncurrentLiabilitiesAbstract
from	ShareholdersEquity
from	RetainedEarnings
from	CapitalStock
from	CapitalSurplus
to	ShareholdersEquityAbstract
from	DeferredGainsOrLossesOnHedges
from	ValuationDifferenceOnAvailableForSaleSecurities
from	ValuationAndTranslationAdjustments
to	ValuationAndTranslationAdjustmentsAbstract

"orgExalge.csv" を
"taxonomyTable.csv" にて振
替変換した結果。



2.11694E+11	HAT	CurrentLiabilitiesAbstract	JPY	2008_03_31	E04081_000
21959460000	NO_HAT	NoncurrentAssetsAbstract	JPY	2008_03_31	E04081_000
400812000	NO_HAT	IntangibleAssetsAbstract	JPY	2008_03_31	E04081_000
4.3491E+11	NO_HAT	CurrentAssetsAbstract	JPY	2008_03_31	E04081_000
2.28635E+11	HAT	BalanceSheetsAbstract	JPY	2008_03_31	E04081_000
44492909000	HAT	NetAssetsAbstract	JPY	2008_03_31	E04081_000
992904000	NO_HAT	ValuationAndTranslationAdjustmentsAbstract	JPY	2008_03_31	E04081_000
89180817000	HAT	ShareholdersEquityAbstract	JPY	2008_03_31	E04081_000
1.84342E+11	HAT	LiabilitiesAbstract	JPY	2008_03_31	E04081_000
1.56989E+11	HAT	NoncurrentLiabilitiesAbstract	JPY	2008_03_31	E04081_000
2.28635E+11	NO_HAT	AssetsAbstract	JPY	2008_03_31	E04081_000

アグリゲートされたバランシート

Aggregated Balance Sheet Data by AADL

2.11694E+11	HAT	CurrentLiabilitiesAbstract	JPY	2008_03_31	E04081_000
21959460000	NO_HAT	NoncurrentAssetsAbstract	JPY	2008_03_31	E04081_000
400812000	NO_HAT	IntangibleAssetsAbstract	JPY	2008_03_31	E04081_000
4.3491E+11	NO_HAT	CurrentAssetsAbstract	JPY	2008_03_31	E04081_000
2.28635E+11	HAT	BalanceSheetsAbstract	JPY	2008_03_31	E04081_000
44492909000	HAT	NetAssetsAbstract	JPY	2008_03_31	E04081_000
992904000	NO_HAT	ValuationAndTranslationAdjustmentsAbstract	JPY	2008_03_31	E04081_000
89180817000	HAT	ShareholdersEquityAbstract	JPY	2008_03_31	E04081_000
1.84342E+11	HAT	LiabilitiesAbstract	JPY	2008_03_31	E04081_000
1.56989E+11	HAT	NoncurrentLiabilitiesAbstract	JPY	2008_03_31	E04081_000
2.28635E+11	NO_HAT	AssetsAbstract	JPY	2008_03_31	E04081_000

~result = 211693995000^<CurrentLiabilitiesAbstract,JPY,2008_03_31,E04081_000>
 +21959460000<NoncurrentAssetsAbstract,JPY,2008_03_31,E04081_000>
 +400812000<IntangibleAssetsAbstract,JPY,2008_03_31,E04081_000>
 +434909857000<CurrentAssetsAbstract,JPY,2008_03_31,E04081_000>
 +228635000000^<BalanceSheetsAbstract,JPY,2008_03_31,E04081_000>
 +44492909000^<NetAssetsAbstract,JPY,2008_03_31,E04081_000>
 +992904000<ValuationAndTranslationAdjustmentsAbstract,JPY,2008_03_31,E04081_000>
 +89180817000^<ShareholdersEquityAbst
 ract,JPY,2008_03_31,E04081_000>+184342000000^<LiabilitiesAbstract,JPY,2008_03_31,E04081_000>
 +156989176000^<NoncurrentLiabilitiesAbstract,JPY,2008_03_31,E04081_000>
 +228635000000<AssetsAbstract,JPY,2008_03_31,E04081_000>

To organization based Economic Simulation

- We can construct organization based world economic simulation model by using XBRL data of companies all over the world.
- We have started the project by using SOARS and AADL.

For example the **Forbes 500** (an annual listing of the **top 500** American companies produced by **Forbes** Magazine) can be used for basic data for world organization based economic simulation.

InfoWorld

[Log-in](#) | [Register](#)

[HOME](#) [NEWS](#) [TEST CENTER](#) [TECHNOLOGIES](#) [BLOGS](#) [AUDIO/VIDEO](#) [EVENTS](#) [AWARDS](#) [NEWSLETTER](#)

SEC moves toward requiring interactive data filings

The SEC is publishing a proposal that would require publically traded companies to submit their reports in the XBRL language

By Grant Gross, IDG News Service
May 14, 2008



[Talkback](#)



[E-mail](#)



[Printer Friendly](#)



[Reprints](#)



[A](#) [A](#)

The U.S. Securities and Exchange Commission (SEC) has taken a major step toward requiring publicly traded companies to submit their reports to the agency in an interactive data format, with backers saying the change will make financial reports easier to analyze.

Related Stories

[IT vendors submit management spec to W3C](#)

Popular Tags

All three SEC members voted to publish a proposal that would require public companies to file reports in eXtensible Business Reporting Language, or XBRL, a programming language related to XML that's being developed by a nonprofit consortium of about 450 companies. Under the proposal, which still needs final approval from the SEC after a public comment period, the transition from text and HTML reports to [XBRL](#) would take three years, with about 500 of the largest U.S. and foreign

DSLとの結合によるモデル記述能力強化

- DSLとの結合
 - (1) エージェントやスポット固有の状態記述のためのDLSとの結合：会計や社会統計の記述言語DSLであるAADL(Algebraic Accounting Description Language)で会計的な状態記述をできるようにする。
 - これにより例えば、Edinet等からクローリングしてきた、企業の会計データを企業をエージェントとしたシミュレーションを容易にする。
 - (2) マクロ変数をエージェント変数に展開する、エージェント展開型としての確率的動学プロセスのモデル記述の作法を明らかにする。
 - エージェントとのマイクロダイナミクスを記述する、数理枠組みをDSLとしそれをSOARS上での記述言語とする

- 社会会計計算の集合論的代数的仕様記述

$$Xq = \sum \left\{ q(e, t_0) < e, VQ, t_0, \# > \mid e \in \Lambda^{PI} \right\}$$

$$= \sum \left\{ \frac{\| \text{Pr}[< e, W, t_0, \# >](Xp) \|}{\| \text{Pr}[< e, PI, t_0, \# >](Xw) \|} < e, VQ, t_0, \# > \mid e \in \Lambda^{PI} \right\}$$

var algeVQSet:ExAlgeSet

= { (na/nb)@<e, "VQ", refTime, "#"> | e<-basePI,
na = norm(proj[<e,"WEIGHT",refTime, "#">](algePI)),
nb = norm(proj[<e,"PI",refTime, "#">](algePI)),
nb != 0 };

var algeVQ:Exalge = sum(algeVQSet);

• そのAADL
による記述

これらをSOARSに組込む

- エージェントダイナミクスの集合論的仕様記述

InfAgent(ActSpot(ω)) : ω と接触する可能性のある患者数

= | { ν | ActSpot(ω) \cap ActSpot(ω) $\neq \emptyset$ 、 $\nu \in \Omega[i](t)$ } |

これはI(t)よりずっと小さい！！

$P[s,i;t](\omega) = \beta \times \text{InfAgent}(\text{ActSpot}(\omega))$ for $\omega \in \Omega[s]$

ご清聴ありがとうございました

Thank You for Your Attention

<http://www.absss.titech.ac.jp/en/>

www.soars.jp