# A Knowledge Module: Buying and Selling

**Joohyung Lee**
Department of Computer Science and Engineering
Arizona State University

**Vladimir Lifschitz**
Department of Computer Sciences
University of Texas at Austin

**Abstract**

This note shows how to formalize a small set of general facts about buying and selling. We begin with summarizing properties of buying/selling informally in English, and give examples of consequences of these assumptions. Then we formalize our assumptions in action language $\mathcal{C}+$ with additive fluents and actions and test the adequacy of the proposed formalization using the Causal Calculator.

## Introduction

This is a "knowledge module" paper written for the AAAI Spring Symposium on Formalizing and Compiling Background Knowledge.[1] It presents a small set of general facts about buying and selling, and shows how to formalize these facts in action language $\mathcal{C}+$ (Giunchiglia *et al.* 2004) with symbols for additive fluents (Lee and Lifschitz 2003) and additive actions (Lee 2005, Section 8.6). The organization of the paper follows the pattern suggested in the Call for Papers:

- a knowledge base written in English;
- examples of its informal consequences;
- a description of the language;
- the formalization;
- a description of testing.

Technically, our formalization is an enhancement of the section of (Lee and Lifschitz 2003) entitled *Reasoning about Money*.

## Knowledge base, in English

(1) A person may own resources of various kinds. One kind is money. Some of the other kinds are items that may be available for sale. Every such item has a price, which is an amount of money.

(2) Buying is a transaction between two persons, the buyer and the seller, and every such transaction involves a certain number of items of some kind that are available for sale.

(3) Buying has the following effects:

(3a) the number of items of the kind involved in the transaction that are owned by the buyer increases; the increase is the same as the number of items involved in the transaction;

(3b) the number of items of the kind involved in the transaction that are owned by the seller decreases by the same quantity;

(3c) the amount of money owned by the seller increases; the increase can be calculated by multiplying the price of the item by the number of items involved in the transaction;

(3d) the amount of money owned by the buyer decreases by the same quantity.

(4) When several transactions are performed simultaneously, the combined effect of these transactions on the amount of any resource owned by a person can be calculated by adding the effects of the individual transactions.

(5) The total amount of money contributed by a person to the transactions in which he currently participates as a buyer does not exceed the amount of money that he owned when these transactions started. The total amount of any resource contributed by a person to the transactions in which he currently participates as a seller of this resource does not exceed the amount of the resource that he owned when these transactions started.

## Examples of informal consequences

*Example 1.* Alice has \$10; how many \$2 roses can she buy? Answer: 5 (or fewer).

*Comments.* This calculation is based on clauses (3d) and (5) of the knowledge base. The calculation would be similar if Alice had \$100; our knowledge base does not permit bulk discounts. Clause (5) is inconsistent with the use of credit cards. The answer in this example depends on the implicit assumption that there are many roses available for sale; this assumption is retracted in Example 5 below.

*Example 2.* As the previous example, except that Alice wants also to buy a \$5 orchid from the same florist.

[1] Call for Papers: `http://www.aaai.org/Symposia/Spring/2006/sssparticipation-06.html#ss05` .

Answer: She has enough money to get 2 roses. She can buy the roses and the orchid at the same time, if she wishes—there is no need to perform one action before the other.

*Example 3.* As the previous example, except that Alice wants to buy the orchid and the roses from two different florists. The same two conclusions can be made.

*Comments.* The conclusion that even in this case the two actions can be performed concurrently depends on fact that the knowledge base does not say anything about the buyer and the seller being at the same place when they conduct the transaction. This conclusion is not unreasonable; we can imagine, for instance, that Alice performs the two transactions by sending both florists an e-mail with individual instructions, or by sending one of her children to get the roses and another to get the orchid. In the next example we include an additional assumption related to locations.

*Example 4.* As in Example 1, Alice has $10 and wants to buy a few $2 roses, except that she can only buy flowers in the store. Her car is currently out of gas, and she needs at least $1 worth of gas to get there. Answer: she'll be able to buy 4 roses, but only after buying gas.

*Comments.* These conclusions depend on some facts about driving that are not included in the knowledge base.

*Example 5.* As in Example 1, Alice has $10 and wants to buy a few roses. Her florist sells them at $2 per stem, but he has only 2 roses left. Another florist has many roses, but he charges $3. Answer: Alice has enough money to buy 4 roses.

*Example 6.* Alice wants to make a Martini, and she has ice at home, but neither gin nor vermouth. A bottle of gin is available at the store for $7, and a bottle of vermouth for $4. Will she succeed? Answer: yes, if she has at least $11.

*Example 7.* Alice wants to buy a new car for $10,000. She has $9,000 and an old car that she can sell to Bob for $2,000. Can she get the new car? Answer: yes, but first she'll have to sell the old car.

*Comments.* Even if Bob happens to be the dealer who is selling the new car to Alice, clause (5) still implies that the two actions cannot be performed concurrently; the knowledge base does not describe trade-ins or exchanges.

## Language of the formalization

The knowledge base above is translated here from English into action language $\mathcal{C}+$ (Giunchiglia *et al.* 2004) with symbols for additive fluents and actions (Lee and Lifschitz 2003), (Lee 2005, Section 8.6), as implemented in the Causal Calculator (CCalc).[2] In the rest of this note we assume some familiarity with this language and with the operation of CCalc.

[2] `http://www.cs.utexas.edu/users/tag/ccalc/` .

```
:- sorts
   agent;
   resource >> item.

:- variables
   Ag,Ag1                    :: agent;
   R                         :: resource;
   It                        :: item;
   N                         :: 1..maxAdditive;
   M                         :: 0..maxAdditive.

:- objects
   money                     :: resource.

:- constants
   price(item,agent,agent)
          :: (0..maxAdditive) + none;
   has(agent,resource)
          :: additiveFluent(0..maxAdditive);
   spending(agent,resource)
          :: additiveAction(0..maxAdditive);
   buy(agent,agent,item)
          :: exogenousAction;
   howMany(agent,agent,item)
          :: attribute(1..maxAdditive)
                 of buy(agent,agent,item).
```

Figure 1: File `buy`, Part 1

## Formalization of the knowledge base

The formalization of the knowledge base is shown in Figures 1–3.

The constant `price` takes three arguments, not just one, because the price of an item, and even its availability for sale, may depend on the seller and the buyer. We did not make `price` a fluent, however—it is not allowed to change. (The knowledge base is not explicit about this.) Accordingly, we will not be able to describe reduced prices and bargaining.

The possible values of `price` are numbers from an initial segment of nonnegative integers and the symbol `none`, which is assigned as the price to the items that are not available for sale. The upper bound `maxAdditive` of the range of prices is not specified in the formalized knowledge base; in the language of CCalc, this is the largest integer allowed in calculations with additive fluents and actions, and its value depends on the queries that we want CCalc to answer.

The fact that `has` is declared as an additive fluent constant corresponds to clause (4) in the knowledge base. An additive fluent is a fluent with numerical values such that the effect of several concurrently executed actions on it can be computed by adding the effects of the individual actions (Lee and Lifschitz 2003).

The additive action constant `spending` is used in Figure 3, as we will see soon, to formalize clause (5). It is similar to the action constant `departing` used in the

```
default price(It,Ag,Ag1)=none.

nonexecutable buy(Ag,Ag,It).
nonexecutable buy(Ag,Ag1,It)
    if price(It,Ag1,Ag)=none.

buy(Ag,Ag1,It) increments has(Ag,It) by N
    if howMany(Ag,Ag1,It)=N.

buy(Ag,Ag1,It) decrements has(Ag1,It) by N
    if howMany(Ag,Ag1,It)=N.

buy(Ag,Ag1,It)
    decrements has(Ag,money) by M*N
    if howMany(Ag,Ag1,It)=N
       & price(It,Ag1,Ag)=M
    where M*N =< maxAdditive.

buy(Ag,Ag1,It)
    increments has(Ag1,money) by M*N
    if howMany(Ag,Ag1,It)=N
       & price(It,Ag1,Ag)=M
    where M*N =< maxAdditive.
```

Figure 2: File buy, Part 2

CCalc formalization of the Missionaries and Cannibals problem with two boats in (Lee 2005, Section 9.2).

The action constant buy takes three arguments: the buyer, the seller, and the kind of items that are being bought. The number of items involved in the transaction is treated as an attribute (Giunchiglia *et al.* 2004, Section 5.6).

The first causal law in Figure 2 says that, by default, an item is assumed to be not available for sale. This default is not part of the knowledge base that we are formalizing, but it turns out to be convenient for describing specific domains. The first of the two causal laws beginning with nonexecutable says that a person cannot buy from himself (not stated explicitly in the English language knowledge base); the second expresses our interpretation of none as the value of price. The last four causal laws in Figure 2 are translations of clauses (3a)–(3d).

The first two causal laws in Figure 3 describe spending(Ag,R) as the total amount of resource R contributed by Ag to all transactions that are being currently executed. The third causal law represents clause (5).

## Testing

The CCalc input file shown in Figure 4 can be used to check that our formalization of the knowledge base leads to the expected consequences in case of Example 1.

Given a $\mathcal{C}+$ action description, a query instructs CCalc to find a path in the corresponding transition

```
buy(Ag,Ag1,It)
    increments spending(Ag1,It) by N
    if howMany(Ag,Ag1,It)=N.

buy(Ag,Ag1,It)
    increments spending(Ag,money) by M*N
    if howMany(Ag,Ag1,It)=N
       & price(It,Ag1,Ag)=M
    where M*N =< maxAdditive.

always spending(Ag,R)=<has(Ag,R).
```

Figure 3: File buy, Part 3

system that satisfies certain conditions, or to establish that there is no such path. In this example, the length maxstep of the path is 1, so that we look here for a single transition. The two equalities following 0: express that this transition begins in a state in which Alice has $10, and that in the course of this transition she buys from the florist 5 roses. There are no restrictions on the number of roses that the florist has, in accordance with default (c).

CCalc finds the following solution, which shows that the action in question can be indeed executed:

```
0:  has(alice,money)=10  has(alice,rose)=0
has(florist,money)=0  has(florist,rose)=12

ACTIONS:  buy(alice,florist,rose,howmany=5)
spending(alice,money)=10
spending(alice,rose)=0
spending(florist,money)=0
spending(florist,rose)=5

1:  has(alice,money)=0  has(alice,rose)=5
has(florist,money)=10  has(florist,rose)=7
```

If we replace 5 in the query with 6 then CCalc will

```
:- maxAdditive :: 12.

:- include 'buy'.

:- objects
  alice,florist      :: agent;
  rose               :: item.

price(rose,florist,Ag)=2.

:- query
maxstep :: 1;
0: has(alice,money)=10,
   howMany(alice,florist,rose)=5.
```

Figure 4: A representation of Example 1

```
:- objects
  alice,florist     :: agent;
  rose,orchid       :: item.

price(rose,florist,Ag)=2.
price(orchid,florist,Ag)=5.

:- query
maxstep :: 1;
0: has(alice,money)=10,
   howMany(alice,florist,orchid)=1,
   howMany(alice,florist,rose)=2.
```

Figure 5: From a representation of Example 2

determine that the problem is unsolvable.

The main part of the formalization of Example 2 is shown in Figure 5. As expected, CCalc finds a solution, and also determines that the problem becomes unsolvable if we replace 2 in the last line of the query by 3. The formalization of Example 3 is similar; we declare `florist1` to be yet another agent, extend the assumptions about prices accordingly, and replace the last line of the query with

```
howMany(alice,florist1,rose)=2.
```

To represent Example 4, we need to include a few assumptions about locations and driving (Figure 6). The causal law beginning with `nonexecutable` says that Alice buys anything only when she is at the same place as the seller. The last four lines of Figure 6 express that initially Alice is at the gas station with $10 in her purse and no gas, and that at a certain later point in time she buys 4 roses. The line `maxstep :: 2..3` instructs CCalc to find a sequence of 2 events satisfying these conditions, and, if this is impossible, try to find a sequence of length 3. CCalc determines that 3 steps are required: buy gas; drive; buy flowers.

The formalization of Example 5 is similar to the formalizations of Examples 1 and 2 above; the condition on the actions to be performed can be expressed in the language of CCalc by the formula

```
howMany(alice,florist,rose)
    +howMany(alice,florist1,rose)=4.
```

In our formalization of Example 6 (Figure 7), the action `makeMartini` does not affect the amount of gin that Alice has, as well as the amounts of vermouth and ice. Recall that the amount of gin is measured here in bottles; we think of a bottle as infinitely large in comparison with the quantity needed for one drink.

The formalization of Example 7, shown in Figure 8, is straightforward. As expected, the goal can be achieved in two steps, but not in one.

## Conclusion

The problem of formalizing properties of buying and selling could be solved here in a rather concise way

```
:- sorts
  location.

:- objects
   alice, gasStationAttendant,
                      florist  :: agent;
   rose,gas                    :: item;
   gasStation, flowerShop      :: location.

:- variables
   L                           :: location.

:- constants
   loc(agent)      :: inertialFluent(location);
   drive(location) :: exogenousAction.

price(rose,florist,Ag)=2.
price(gas,gasStationAttendant,Ag)=1.

caused loc(gasStationAttendant)=gasStation.
caused loc(florist)=flowerShop.

nonexecutable buy(alice,Ag,It)
    if loc(alice)\=loc(Ag).

drive(L) causes loc(alice)=L.
nonexecutable drive(L) if has(alice,gas)=0.

:- query
maxstep :: 2..3;
0: loc(alice)=gasStation,
   has(alice,money)=10,
   has(alice,gas)=0;
(maxstep-1): howMany(alice,florist,rose)=4.
```

Figure 6: From a representation of Example 4

because of several useful features of language $\mathcal{C}+$ and its CCalc implementation. This language incorporates a built-in solution to the frame problem (`inertialFluent` declarations). Its "action language" constructs, such as `causes` and `nonexecutable`, allow us to avoid talking about time explicitly (except in queries). In $\mathcal{C}+$ we can declare non-Boolean fluents, and in particular number-valued fluents, such as `has`. A number-valued fluent can be declared additive, and effects of actions on additive fluents can be described using `increments` propositions.

The availability of additive fluent and action constants was particularly valuable in this experiment, because the English-language knowledge base that we started with included assumptions about the concurrent execution of buying/selling actions. Additive fluents can be described also in the situation calculus (Erdem and Gabaldon 2005).

```
:- objects
  alice,store              :: agent;
  ice,gin,vermouth,martini :: item.

:- constants
  makeMartini              :: exogenousAction.

price(gin,store,Ag)=7.
price(vermouth,store,Ag)=4.

nonexecutable makeMartini
  if has(alice,gin)=0 ++ has(alice,vermouth)=0
     ++ has(alice,ice)=0.

makeMartini increments has(alice,martini) by 1.

:- query
maxstep :: 1..2;
0: has(alice,money)=11,
   has(alice,ice)>0,
   has(alice,gin)=0,
   has(alice,vermouth)=0;
(maxstep-1): makeMartini.
```

Figure 7: From a representation of Example 6

```
:- objects
  alice,bob,dealer          :: agent;
  oldCar,newCar             :: item.

price(oldCar,alice,bob)=2.
price(newCar,dealer,alice)=10.

:- query
maxstep :: 1..2;
0: has(alice,money)=9,
   has(alice,oldCar)=1,
   has(alice,newCar)=0;
maxstep:
   has(alice,newCar)=1.
```

Figure 8: From a representation of Example 7

## Acknowledgements

We are grateful to Michael Gelfond for suggesting that an example of automated reasoning about money that we have published earlier can be extended to a knowledge module. Thanks to Selim Erdǒgan, Paolo Ferraris, Wanwan Ren and anonymous referees for comments on a draft of this note. The second author was partially supported by the National Science Foundation under Grant IIS-0412907.

## References

Esra Erdem and Alfredo Gabaldon. Cumulative effects of concurrent actions on numeric-valued fluents. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 627–632, 2005.

Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1–2):49–104, 2004.

Joohyung Lee and Vladimir Lifschitz. Describing additive fluents in action language $\mathcal{C}+$. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1079–1084, 2003.

Joohyung Lee. *Automated Reasoning about Actions.*[3] PhD thesis, University of Texas at Austin, 2005.

[3]http://www.cs.utexas.edu/users/appsmurf/papers/dissertation
.