Evolving Contracts

Gilles Barthe
IMDEA Software, Madrid

September 2009

(Work in progress with Gordon Pace and Gerardo Schneider)

Motivations: contracts everywhere

1. Conventional contracts (legal, commercial, etc)

Motivations: contracts everywhere

1. Conventional contracts (legal, commercial, etc)
2. "Programming by contract" or "Design by contract" (e.g., Eiffel)
    - Relation between pre- and post-conditions of routines, method calls, invariants, temporal dependencies, etc

Motivations: contracts everywhere

1. Conventional contracts (legal, commercial, etc)
2. “Programming by contract” or “Design by contract” (e.g., Eiffel)
   - Relation between pre- and post-conditions of routines, method calls, invariants, temporal dependencies, etc
3. Service-Level Agreement (web services and SOA)

Motivations: contracts everywhere

1. Conventional contracts (legal, commercial, etc)
2. "Programming by contract" or "Design by contract" (e.g., Eiffel)
   - Relation between pre- and post-conditions of routines, method calls, invariants, temporal dependencies, etc
3. Service-Level Agreement (web services and SOA)
4. Behavioral interfaces
   - Specify the sequence of interactions between participants

Motivations: contracts everywhere

1. Conventional contracts (legal, commercial, etc)
2. “Programming by contract” or “Design by contract” (e.g., Eiffel)
   - Relation between pre- and post-conditions of routines, method calls, invariants, temporal dependencies, etc
3. Service-Level Agreement (web services and SOA)
4. Behavioral interfaces
   - Specify the sequence of interactions between participants
5. Security policies

Motivations: contracts everywhere

1. Conventional contracts (legal, commercial, etc)
2. "Programming by contract" or "Design by contract" (e.g., Eiffel)
    - Relation between pre- and post-conditions of routines, method calls, invariants, temporal dependencies, etc
3. Service-Level Agreement (web services and SOA)
4. Behavioral interfaces
    - Specify the sequence of interactions between participants
5. Security policies
6. Contracts in virtual organisations, multi-agent systems, etc

Evolvability

- Principals evolve
- Systems evolve
- Contracts should evolve

- Most works on contracts only deal with static contracts
- One exception: administrative role-based access control

Goals

- Long-term goal: develop a unifying theory of contracts, covering different issues such as:
  - contract generation, composition and *evolution*
  - contract analysis and verification
  - first-class contracts
- Short-term goals:
  - model dynamicity of contracts: spillover, power
  - provide enforcement mechanisms for dynamic contracts
  - study the relationship and correctness of the distinct enforcement mechanisms

Contracts, informally

- Agreements between a system involving two or more parties
  - asymmetric: user and provider, producer and consumer, etc.
  - symmetric: participants in social networks, P2P networks, etc
- Regulate actions that a system may undertake
- Static contracts are fully determined at onset.

- We are interested in enforceable contracts:
  - ie contracts can be validated against a reference semantics (of contracts and systems).
- For the purpose of this talk, one can think of a system as a (possibly distributed) program in which code from different principals are executed.

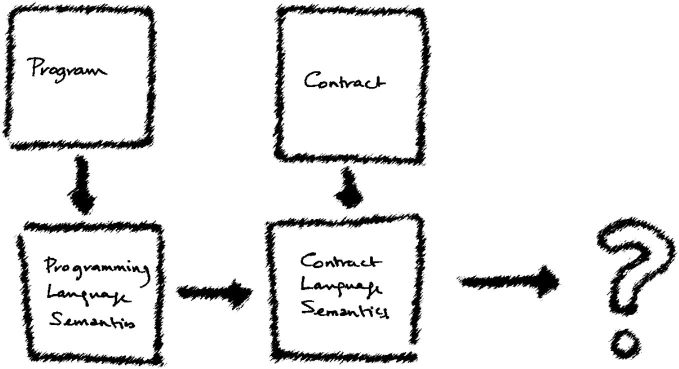
Focus

- Formalize enforcement mechanisms for contracts
- Prove/disprove equivalence between mechanisms

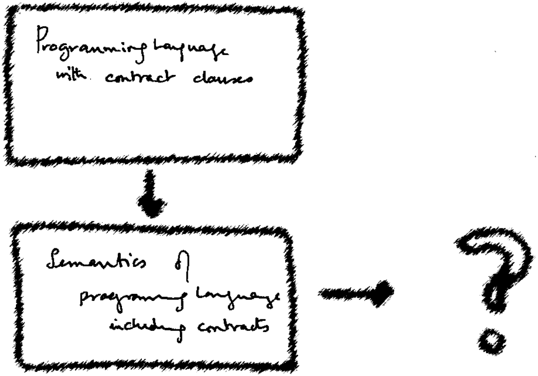
Plan

- Start with static contracts
- Extend to dynamic contracts

Verifying Static Contracts



Verifying Dynamic Contracts



Setting – dimensions of choice

- What is a system and which modelling language to use?
- Which contract language (syntax and semantics) to use?
- When does a system comply with a contract?

Each dimension is a topic on its own.

Setting – formalisation

- Systems are modelled as sequences of actions. Formally, we consider a set A of actions and let $\mathit{Trace} = A^\star$.
- Contracts are modelled as predicates over systems. Formally, we consider a set $\mathit{Contract}$ of contracts, and a relation $\vdash_{\mathrm{offl}} \subseteq \mathit{Trace} \times \mathit{Contract}$.

- Pros: very general and applicable.
  - Captures essential issues of contract enforcement
  - Abstracts away the specifics of formalisms for contracts and systems
- Cons:
  - Time escapes the model, among many other parameters
  - Hyperproperties are not considered

There is still a story to tell. . .

Motivating example: DRM (after Barth and Mitchell)

- Consider a scenario with an online music store using DRM.
- Users receive rights to play digital songs.
- Alice may pay to obtain the permission to listen to songs a, b, and c for a total of ten times.
- For being a loyal customer, the following day, the store decides to provide Alice with a promotion to either play song d once before the end of the month, or to play either song d or a once before the end of the month.

Motivating example: DRM (after Barth and Mitchell)

- Alice obtains the right to listen to songs *a*, *b* and *c* for a total of ten times.
- Alice plays song *a* twice, *b* four times, and *c* three times, for a total of nine plays.
- The DRM agent in her music player decrypts the songs, allows Alice to play the songs, and notes that she has one play remaining.
- The following day, Alice receives another promotion — she is offered the choice of two rights:
  - the right to play song *d* once; or
  - the right to play either song *a* or song *d* once.

  She opts for the second right because she reasons that it is more flexible.

DRM: Choosing rights and the right to choose

- The rights Alice now possesses are:
  - Play either song a, b, or c (acquired the first day).
  - Play either song a or d before the end of the month.
- If she now listens to song a, which right should the DRM manager opt to strike out?
- If it strikes out the second (as the more restrictive), she will not be able to listen to d later on.
- But if she had chosen the more restrictive promotion, the second option would have been struck out, and she would have been able to listen to song d later on.
- In fact: no online algorithm can be complete.
- The problem is due to non-atomic rights.

Verification of static contracts: the big picture

We define three enforcement mechanisms

- Offline verification
- Online verification
- History-based verification
- State-based verification

and provide conditions under which they coincide.

Online verification

Contract is updated after every action:

$$step \in A \to Contract \to Contract_{\perp}$$

(*step* is undefined whenever the action violates the contract)

Satisfaction

$$\begin{array}{rcll} \langle\rangle \vdash_{dyn} C & \stackrel{df}{=} & \langle\rangle \vdash_{\mathrm{offl}} C & \\ a :: t \vdash_{dyn} C & \stackrel{df}{=} & t \vdash_{dyn} step(a, C) & \text{if } step(a, C) \neq \perp \\ a :: t \vdash_{dyn} C & \stackrel{df}{=} & \text{false} & \text{if } step(a, C) = \perp \end{array}$$

Soundness and completeness of online verification

Let C range over contracts, t over traces, and a over actions.

Soundness

$$t \vdash_{\mathrm{offl}} \mathit{step}(a, C) \Rightarrow a :: t \vdash_{\mathrm{offl}} C$$

If *step* is sound, then for every contract C and trace t

$$t \vdash_{\mathrm{onl}} C \Rightarrow t \vdash_{\mathrm{offl}} C$$

Completeness

$$t \vdash_{\mathrm{offl}} \mathit{step}(a, C) \Leftarrow a :: t \vdash_{\mathrm{offl}} C$$

If *step* is complete, then for every contract C and trace t

$$t \vdash_{\mathrm{onl}} C \Leftarrow t \vdash_{\mathrm{offl}} C$$

Application: simple language of rights and obligations

- $A = R \sqcup O$
- $Contract = R^+ \times O^+$ (multisets as conjunctions)
- $t \vdash_{\text{offl}} (C_r, C_o)$ iff $C_o \subseteq t_o$ and $t_r \subseteq C_r$
- where t_o and t_r are the projections of t over obligations and rights.

Step function

$$step(a, (C_r, C_o)) = \begin{cases} (C_r \setminus \{a\}, C_o) & \text{if } a \in R \wedge a \in C_r \\ \bot & \text{if } a \in R \wedge a \notin C_r \\ (C_r, C_o \setminus \{a\}) & \text{if } a \in O \end{cases}$$

(Note that $\setminus$ is defined for multisets of actions, therefore removes at most one occurence of an action from a multiset of actions.)

step is sound and complete.

Non-application: non-atomic rights

- Rights are disjunctions of atomic rights (a la DRM)
- $\mathit{Contract} = (R^+)^+ \times O^+$ (multisets as conjunctions)
- $t \vdash_{\mathrm{offl}} (C_r, C_o)$ iff $C_o \subseteq t_o$ and $t_r \sqsubseteq C_r$ where:
  - t_o and t_r are the projections of t over obligations and rights
  - $l \sqsubseteq l'$ if there exists $l_0 \subseteq l'$ such that $l \leq l'$ pointwise.

Step function

$$\mathit{step}(a, (C_r, C_o)) = \begin{cases} (C_r \setminus \{b\}, C_o) & \text{if } a \in R \wedge a \leq b \wedge b \in C_r \\ \perp & \text{if } a \in R \wedge a \notin C_r \\ (C_r, C_o \setminus \{a\}) & \text{if } a \in O \end{cases}$$

(Note that *step* is not a function.)

No function whose graph is included in *step* is sound and complete.

History-based verification

Contract is updated after a set of actions, and only when it does not restrict future (legitimate) choices of users.

$$\textit{h-step} \in \textit{Trace} \times \textit{Contract} \rightarrow (\textit{Trace} \times \textit{Contract})_{\perp,\mathrm{void}}$$

h-step returns void if the trace/contract pair is not updated.

Satisfaction (simplified definition)

$$\begin{array}{rcll} t \vdash_{hist} C & \stackrel{df}{=} & t \vdash_{\mathrm{offl}} C & \text{if } \textit{h-step}(t, C) = \mathrm{void} \\ t_0 \frown t \vdash_{hist} C & \stackrel{df}{=} & t' \frown t \vdash_{hist} C' & \text{if } \textit{h-step}(t_0, C) = (t', C') \\ t_0 \frown t \vdash_{hist} C & \stackrel{df}{=} & \text{false} & \text{if } \textit{h-step}(t_0, C) = \perp \end{array}$$

Generalizes offline verification (always map to void), and online verification (always update based on first element of the list).

Soundness and completeness of history-based verification

Let C range over contracts, t over traces, and a over actions.

Soundness

$$t' \frown t \vdash_{\mathrm{offl}} C' \wedge \textit{h-step}(t_0 \frown t, C) = (t', C') \Rightarrow t_0 \frown t \vdash_{\mathrm{offl}} C$$

If *h-step* is sound, then for every contract C and trace t

$$t \vdash_{hist} C \Rightarrow t \vdash_{\mathrm{offl}} C$$

Completeness

$$\left.\begin{array}{l} t' \frown t \vdash_{\mathrm{offl}} C' \wedge \textit{h-step}(t_0 \frown t, C) = (t', C') \\ \vee \\ \textit{h-step}(t_0 \frown t, C) = \mathrm{void} \end{array}\right\} \Leftarrow t_0 \frown t \vdash_{\mathrm{offl}} C$$

If *step* is complete, then for every contract C and trace t

$$t \vdash_{hist} C \Leftarrow t \vdash_{\mathrm{offl}} C$$

Application: non-atomic rights

One can define sound and complete history-based enforcement for non-atomic rights. Many strategies:

- Remove sets of rights as soon as possible

$$\text{remove} : R^+ \times (R^+)^+ \rightharpoonup R^+ \times (R^+)^+$$

(w/o compromising soundness)

Step function

$$h\text{-}step(t, (C_r, C_o)) = \begin{cases} (C_r, C_o \setminus \{a\}) & \text{if } a \in O \\ (t'_r, (C'_r, C_o)) & \text{if } t = a :: t_0 \wedge a \in R \wedge \\ & \text{remove}(t_r, C_r) = (t'_r, C'_r) \\ \ldots & \end{cases}$$

- One can try to remove sets of rights at regular intervals
- One never removes sets of rights (offline verification is a special case)

Back to DRM

- Recall the DRM system, in which Alice had a contract C, is a set of disjunctions of permissions to listen to songs.
- The history h is a multiset of rights that were consumed but not yet deducted from the contract.
- When Alice listens to a song s:
  - if there is a deterministic way of reducing the current contract C with history $s :: h$ then reduce it and remove the relevant rights from C, and update history;
  - otherwise simply extend history with s.
- For example, with contract $(a \vee b) \wedge (a \vee c) \wedge (b \vee d)$, upon hearing song a, we do not know which clause to remove, so we don't remove any. If we then receive a c, we can now use up both the a and c to reduce the contract to get $b \vee d$.

Dynamic contracts

- Contracts are sets of clauses
- Clauses can be enacted or withdrawn throughout execution
- We consider an extended set of actions with specific actions for enacting and withdrawal of clauses

Dynamic contracts

- Contracts are sets of clauses
- Clauses can be enacted or withdrawn throughout execution
- We consider an extended set of actions with specific actions for enacting and withdrawal of clauses

New phenomenae

- What happens if a clause is withdrawn?
- Who enacts or withdraws clauses

Dynamic contracts

- Contracts are sets of clauses
- Clauses can be enacted or withdrawn throughout execution
- We consider an extended set of actions with specific actions for enacting and withdrawal of clauses

New phenomenae

- What happens if a clause is withdrawn?
- Who enacts or withdraws clauses

New notions

- Spillover: what happens if a clause is withdrawn?
- Power: who enacts or withdraws clauses

Dynamic contracts

- Contracts are sets of clauses
- Clauses can be enacted or withdrawn throughout execution
- We consider an extended set of actions with specific actions for enacting and withdrawal of clauses

New phenomenae

- What happens if a clause is withdrawn?
- Who enacts or withdraws clauses

New notions

- Spillover: what happens if a clause is withdrawn?
- Power: who enacts or withdraws clauses

Warning

- Work in progress.
- Need to instantiate new notions to multiple languages.

What is spillover?

When a clause is withdrawn, some parts of it may spillover beyond its termination. Examples:

- For the coming week, if you pay for a song, you will get an extra song download for free.
- If you buy three songs, you get an extra one for free.
- You are allowed to transfer resources tax-free to another player, but if you leave the group before they are delivered, you will have pay taxes on them.

Extended setting

New actions to enact and withdraw a contract

$$\overline{A} ::= A \mid enact(Loc, Contract) \mid withdraw(Loc, Loc)$$

(second location in withdrawal used for spillover)

Traces

Subject to well-formedness conditions. Should never:

- contain two enactments at the same location;
- withdraw twice at the same location;
- update a contract at an already active location;
- withdraw a clause that was not enacted or present at the onset of executions;
- set up a spillover clause through a withdrawal at a location in use.

Active and consumed locations

Start from a set L_0 of initially active locations.

$$\begin{array}{rcl} ActLoc_{L_0}(\langle\rangle) & = & L_0 \\ ActLoc_{L_0}(enact(n,c) :: t) & = & \{n\} \cup ActLoc_{L_0}(t) \\ ActLoc_{L_0}(withdraw(n,n') :: t) & = & \{n'\} \cup ActLoc_{L_0}(t) \setminus \{n\} \\ ActLoc_{L_0}(a :: t) & = & ActLoc_{L_0}(t) \end{array}$$

where in the last clause it is assumed that a neither enacts nor withdraws an action.

$$\begin{array}{rcl} ConsLoc_{L_0}(\langle\rangle) & = & L_0 \\ ConsLoc_{L_0}(enact(n,c) :: t) & = & \{n\} \cup ConsLoc_{L_0}(t) \\ ConsLoc_{L_0}(withdraw(n,n') :: t) & = & \{n'\} \cup ConsLoc_{L_0}(t) \\ ConsLoc_{L_0}(a :: t) & = & ConsLoc_{L_0}(t) \end{array}$$

Every active location is also consumed.

Extracting subtraces wrt consumed locations

Let n be a consumed location in t. The active subtrace of t for n is $G_{L_0}(t, n) = \mathit{reduce}(t')$ where t' is uniquely determined by the decomposition:

$$\begin{array}{rcl} t & = & t' \frown \mathit{withdraw}(n, n') \frown t_1 \\ t & = & t_0 \frown \mathit{enact}(n, c) \frown t' \frown \mathit{withdraw}(n, n') \frown t_1 \\ t & = & t_0 \frown \mathit{enact}(n, c) \frown t' \\ t & = & t_0 \frown \mathit{withdraw}(n', n) \frown t' \frown \mathit{withdraw}(n, n'') \frown t_1 \\ t & = & t_0 \frown \mathit{withdraw}(n', n) \frown t' \end{array}$$

where in the first case it is implicitly assumed that $n \in L_0$ and in the third and fifth cases that $\mathit{withdraw}(n, n') \notin t'$.

Initial clause of a location

- Let $spillover : A^* \times Contract \rightarrow Contract$.
- Let $F : L_0 \rightarrow Contract$.
- Define $\hat{F} : \forall t,\ ConsLoc_{L_0}(t) \times \overline{A}^* \rightarrow Contract$.

$$\hat{F}(t,n) = \begin{cases} F(n) & \text{if } \ n \in L_0 \\ c & \text{if } \ t = t_0 \frown enact(n,c) \frown t' \\ spillover(t', \hat{F}(t', n')) & \text{if } \ t = t_0 \frown withdraw(n', n) \frown t_1 \\ & \ \text{and } t' = G_{L_0}(t,n) \end{cases}$$

Contract satisfaction

Let t be a trace and C be a contract mapping.

- Local validity for a location n: $t \,\overline{\vdash}_{\mathit{offl}}\, (n, C)$ iff $G_{L_0}(t, n) \vdash_{\mathrm{offl}} \hat{F}(t, n)$
- Global validity: $t \,\overline{\vdash}_{\mathit{offl}}\, C$ iff $t \,\overline{\vdash}_{\mathit{offl}}\, (n, C)$ for every $n \in \mathit{ConsLoc}_{L_0}(t)$.

Verification of dynamic contracts: the big picture

As before: we define three enforcement mechanisms

- Offline verification
- Online verification
- History-based verification
- State-based verification

and provide conditions under which they coincide.

Power

- Why do we need power?
  - To model who can modify contracts and how
- The roads to power (in progress)
  - Use contracts to control contract evolution
  - Extend the set of actions
  - Reuse previous results on satisfaction and enforcement mechanisms
- Using power (to do)
  - cast existing frameworks for evolving security policies as power
  - more examples

- Consider a set of principals, and assign principals to actions
- Specify rights and prohibitions of principals
- Can be checked using a static check function, inducing just side conditions to the definitions in the semantics

But rights and prohibitions of principals are static!

Models of power: Permissions, Prohibitions and Power

- Add operators for permission $\mathcal{P}(a)$ and prohibition $\mathcal{F}(a)$ to do an action a, in the contract language.
- Principal p has permission to write contract clause c at location n can be written as $W_p(c, a) = \mathcal{P}(enact_p(c, n))$, and prohibition to do so as $N_p(c, a) = \mathcal{F}(enact_p(c, n))$.
- Allow reasoning about power *within* the model itself.
- This can be used to model power delegation.
- Negotiation can also be modelled in this manner, eg by adding rights to change a contract, until the point of agreement, upon which the contract is frozen — all rights to change it are withdrawn.

Conclusions

- Preliminary exploration of dynamic contracts
- Captured two new ideas: spillover and power
- Instantiations to more complex contract languages are required
- Only a first step towards first-class contracts
- Next step is to embed a contract language in a programming language. Hopefully modular in the contract language via a suitable API.