3. A Type System Equivalent to Higher-Order Model Checking

Goal: Give a type-based algorithm for solving HOMC.

More precisely,

given APTA A, construct a type system $\mathcal{T}_A$

so that for a scheme G we have

G is well-typed in $\mathcal{T}_A$ iff $[\![G]\!]$ is accepted by A.

Result due to

Kobayashi & Ong LICS '09.

Advantages over Ong's LICS '06 algorithm:

Simplicity: Correctness follows from two arguments,

↳ Correctness of the type system

↳ Correctness of the type-checking algorithm.

Complexity: If the automaton is fixed

and the size of types are bounded by a constant,

the algorithm runs in time linear

in the size of the recursion scheme.

Ong's algorithm still needs n-EXPTIME.

Flexibility: The algorithm can be modified to deal with extensions

of recursion schemes:

↳ Polymorphism: $(o \to o) \wedge ((o \to o) \to (o \to o))$.

↳ Finite data domains.

Technology: From type-theoretic point of view,

type system has interesting new features:

↳ Flags and priorities express

when a variable can be used

↳ Well typedness is by winning a parity game.

2.1 Types

Definition:

Consider APTA $A = (Q, \Sigma, \delta, q_I, \Omega)$.

The set of atomic types Θ and the set of types τ are defined by simultaneous induction:

$$\Theta ::= q \mid \underbrace{\tau}_{\text{type}} \to \Theta \qquad \text{// atomic types}$$

$$\tau ::= \bigwedge \{ (\underbrace{\Theta_1}_{\text{atomic type}}, m_1), \ldots, (\Theta_k, m_k) \},$$

where $q \in Q$ and $m_1, \ldots, m_k \in Range(\Omega)$.

Notation:

- We write $(\Theta_1, m_1) \wedge \ldots \wedge (\Theta_k, m_k)$ or $\bigwedge_{i=1}^{k} (\Theta_i, m_i)$ for $\bigwedge \{ (\Theta_1, m_1), \ldots, (\Theta_k, m_k) \}$.
- We write $\top$ for $\bigwedge \emptyset$.
- We extend the priorities to all atomic types by $\Omega(\tau \to \Theta) := \Omega(\Theta)$.

Intuition:

- Type $(q_1, m_1) \wedge \ldots \wedge (q_k, m_k) \to q$ describes a function
  - ↳ that takes a tree which can be accepted from q_1 and from q_2 and ... and from q_k,
  - ↳ and returns a tree that is accepted from state q.
  - ↳ The priority m_i describes the maximal priority on the path

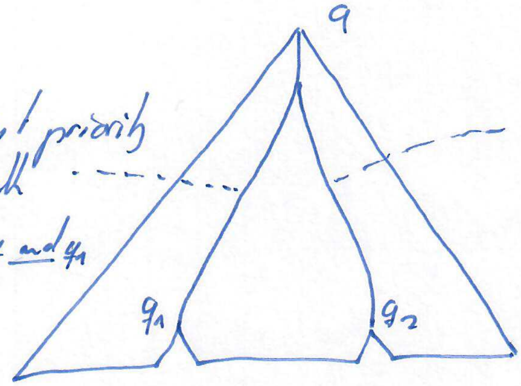
from the root of the output tree (of type q)
to the root of the input tree (of type q_i).

<u>Hence:</u> The input tree can be used as a tree of type q_i
only <u>after</u> visiting a state of priority m_i and
<u>before</u> visiting a state of priority $> m_i$.

<u>Illustration:</u>

The type $(q_1, m_1) \wedge (q_2, m_2) \to q$
describes the following tree function:

The largest priority on this path <u>including</u> q <u>and</u> q_1 is m_1.



The largest priority on this path <u>including</u> q <u>and</u> q_2 is m_2.

So far, the set of types is not related to <u>kinds</u>.
Define well-formed types via two relations:

$\tau :: k$ = "τ is of kind k".

$\theta ::_a k$ = "θ is an atomic type of kind k".

<u>Definition:</u>

- The relations $::$ and $::_a$ are the least relations satisfying the following rules:

$$\frac{}{q ::_a o} \qquad \frac{\tau :: k_1 \quad \theta ::_a k_2}{\tau \to \theta ::_a k_1 \to k_2} \qquad \frac{\theta_i ::_a k \text{ for all } 1 \leq i \leq n}{\wedge\{(\theta_1, m_1), \dots, (\theta_n, m_n)\} :: k}$$

- A type τ (and an atomic type θ) is <u>well-formed</u>, if

(1) $\tau :: k$ (resp. $\Theta ::_a k$), for some kind k

(2) for each subexpression $\bigwedge_{i=1}^{k} (\Theta_i, m_i) \to \Theta'$, we have

$$m_i \geq \max\{\Omega(\Theta_i), \Omega(\Theta')\}, \text{ for all } 1 \leq i \leq k.$$

Example:

- $q_1 \wedge ((q_2, 1) \to q_3)$ is not well-formed, combines types of different kinds.
- $(q_1, m_1) \wedge (q_2, m_2) \to q$ is well-formed, if

$$m_1 \geq \max\{\Omega(q_1), \Omega(q)\}$$
$$m_2 \geq \max\{\Omega(q_2), \Omega(q)\}.$$

This reflects the fact that m_1 and m_2 are the largest priorities on the above paths, including the root and the leaves.

$\boxed{\text{From now on, we only consider well-formed types.}}$

2.2 Type Environments and Type Judgements

Definition:

- A flagged type is an expression $(\Theta, m)^b$ with $b \in \{t, f\}$. We use σ for flagged types.
- A type environment Γ is a set of bindings $x : \sigma$.

With this, type judgements will have the form

$$\Gamma \vdash t : \Theta,$$

where t is a term and where non-terminals are treated
as variables that are bound in Γ.

Note that Γ uses flagged types but
t receives a normal (well-formed) atomic type.

Explanation:

- Γ may contain multiple occurrences of the same variable.
- Each atomic type of a variable is annotated by a flag.

The flag indicates when the variable can be used
as a value of that type:

$x : (q,m)^{\varepsilon} \in \Gamma$ means

x can only be used
before visiting a state of priority $> m$.

$x : (q,m)^{f} \in \Gamma$ means

x can only be used
before visiting a state of priority $> m$ and
after visiting a state of priority m.

Hence, if $x:(q,m)^{f} \in \Gamma$, then the largest priority
on the path from the current node
to the node where x is used equals m.

We have not yet defined the set of type judgments
(that we consider valid).
The following examples are meant to develop some intuition
to which type judgements should be valid.

Example: Let $\Omega(q) = 0$.

(1) $\{ x : (q, 1)^f \} \vdash x : q$ should be <u>invalid</u>.

The type environment says that x can be used only after visiting a state of priority 1 in the context. But the only state that has been visited in the context is the one from which x is accepted, namely q. Since q has priority 0, x can not be used.

(2) $\{ x : (q, 1)^t \} \vdash x : q$ should be <u>valid</u>.

Since the flag is t, x can be used any time before a priority larger 1 is seen.

(3) $\{ x : (q, 1)^f, \ y : ((q, 1) \to q, 0)^f \} \vdash y\,x : q$ should be <u>valid</u>.

Function y uses the argument x only after visiting a state of priority 1. Note that $y\,x$ expects a context where the highest priority is 0. The context only consists of q, which meets the requirement.

(4) $\{ x : (q, 0)^f, \ y : ((q, 1) \to q, 0)^f \} \vdash y\,x : q$ should be <u>invalid</u>.

The flagged type $(q, 0)^f$ of x requires that the largest priority seen before using x is 0. But y uses x in a context where a state of priority 1 is visited.

Notation:

- We drop set braces and write
$$\Gamma, x : \bigwedge_{i=1}^{k} (\theta_i, m_i)^{b_i} \quad \text{for} \quad \Gamma \cup \{ x : (\theta_1, m_1)^{b_1}, \ldots, x : (\theta_k, m_k)^{b_k} \},$$
where x is assumed not to occur in Γ.
This in particular means that type environments are read conjunctively — all the constraints hold.

- We assume <u>injectivity</u> on flags,

  if $x:(\Theta,m)^b$, $x:(\Theta,m)^{b'} \in \Gamma$, then $b=b'$.

  This frees us from having to give a meaning to

$$x: (\Theta,m)^t \wedge (\Theta,m)^f,$$

  which has two sensible interpretations:

  <u>pessimistic</u>: x can only be used after m has been seen,

  so $x: (\Theta,m)^t \wedge (\Theta,m)^f$

  is equivalent to $x:(\Theta,m)^f$.

  <u>optimistic</u>: x can be used already before m,

  so

$$x: (\Theta,m)^t \wedge (\Theta,m)^f$$

  is equivalent to $x:(\Theta,m)^t$.

The type system will somehow implement the latter interpretation.
To this end, it uses the following lifting $\uparrow$ of flagged types.

- The <u>lifting of flagged types</u> tracks the occurrence of priorities and flips flags f to t:

$$(\Theta,m)^b \uparrow m' := \begin{cases} (\Theta,m)^b & \text{if } m' < m \\ (\Theta,m)^t & \text{if } m' = m \\ \text{undef.} & \text{if } m' > m. \end{cases}$$

We generalize the function to type environments,

$$\{x_1:\sigma_1, \dots, x_n:\sigma_n\} \uparrow m := \{x_1:\sigma_1\uparrow m, \dots, x_n:\sigma_n\uparrow m\}.$$

Flagged types that turn to undef are removed from the environment.
Consider now the situation where

$$x:(\Theta,m)^b \in \Gamma$$

and we execute the check

$$(\theta, m)^b \uparrow m' = (\theta, m)^t.$$

It states that x can be used if

$$b = t \text{ and } m' \leq m$$

or $$b = f \text{ and } m' = m.$$

- We write $dom(\Gamma)$ for the set $\{x \mid \exists \theta, m, b : x : (\theta, m)^b \in \Gamma\}$.

We give the rules for deriving type judgements and afterwards add the explanation.

Definition:

The set of type judgements $\Gamma \vdash t : \theta$ is defined by induction over the following four rules:

$$(\text{T-VAR}) \quad \frac{(\theta, m)^b \uparrow \Omega(\theta) = (\theta, m)^t}{x : (\theta, m)^b \vdash x : \theta.}$$

$$(\text{T-CONST}) \quad \frac{\{(i, q_{ij}) \mid 1 \leq i \leq n,\ 1 \leq j \leq k_i\} \text{ satisfies } \delta_A(q, a)}{\emptyset \vdash a : \bigwedge_{j=1}^{k_1} (q_{1,j}, m_{1,j}) \to \dots \to \bigwedge_{j=1}^{k_n} (q_{n,j}, m_{n,j}) \to q,}$$

Here $m_{i,j} := \max\{\Omega(q_{i,j}), \Omega(q)\}$.

$$(\text{T-APP}) \quad \frac{\Gamma_0 \vdash t_0 : (\theta_1, m_1) \wedge \dots \wedge (\theta_k, m_k) \to \theta \qquad \Gamma_i \uparrow m_i \vdash t_1 : \theta_i \text{ for all } 1 \leq i \leq k}{\Gamma_0 \cup \Gamma_1 \cup \dots \cup \Gamma_k \vdash t_0\, t_1 : \theta.}$$

$$(\text{T-ABS}) \quad \frac{\Gamma, x : \bigwedge_{i \in I} (\theta_i, m_i)^f \vdash t : \theta \qquad I \subseteq J}{\Gamma \vdash \lambda x.t : \bigwedge_{i \in J} (\theta_i, m_i) \to \theta.}$$

Comments:

(T-VAR) Since x has no future context, the only state that is visited in the context is the rightmost one in Θ. The corresponding priority $\Omega(\Theta)$ is taken into account when checking whether x can be used (with the above lifting).

(T-CONST) The premise means that, when reading a, the automaton can spawn states q_{ij} and read the ith subtree from state q_{ij}. Thus, for $a\, t_1 \ldots t_n$ to have type q (be accepted from state q) it is sufficient that t_i has type q_{ij} for all $j \in \{1, \ldots, k_i\}$.

(T-APP) The first premise requires that the argument of t_0 should have types $\Theta_1, \ldots, \Theta_k$. The second premise requires that t_1 has these types. Furthermore, the first premise means that the argument t_1 is used as a value of type Θ_i only in a context where the least priority (seen since function t_0 is called) is m_i. Operation $T_i \uparrow m_i$ takes this into account.

(T-ABS) Strengthening the requirements ($I \subseteq J$) is allowed. Moreover, the bindings on x are annotated by l, meaning that x can only be used after the expected priority has been seen. It is the proof that has to make sure the priority occurs.

Example:

Consider $\Sigma = \{ a : o \to o \to o,\ b : o \to o,\ c : o \}$

and $\mathcal{A}_1 = (\Sigma, \{q_0, q_1\}, \delta_1, q_0, \underbrace{\{q_0 \mapsto 2, q_1 \mapsto 1\}}_{\Omega_1})$,

where for each $q \in \{q_0, q_1\}$:

$$\delta_1(q, a) = (1, q) \wedge (2, q)$$

$$\delta_1(q, b) = (1, q_1)$$

$$\delta_1(q, c) = \text{true}.$$

By Rule (T-CONST), we obtain

$a : \Theta_a$ with $\Theta_a = (q_0, 2) \to (q_0, 2) \to q_0$

$a : (q_1, 1) \to (q_1, 1) \to q_1$

$b : (q_1, 2) \to q_0$

$b : (q_1, 1) \to q_1$

$c : q_0$

$c : q_1$.

Let $\Theta = (q_0, 2) \wedge (q_1, 2) \to q_0$

$\Gamma_1 = \{ F : (\Theta, 2)^\varepsilon \} \cup \Gamma_2$ with $\Gamma_2 = \{ x : (q_1, 2)^\varepsilon \}$.

Then for $q_i \in \{q_0, q_1\}$, we have

$$\dfrac{\dfrac{}{\emptyset \vdash b : (q_1, \Omega_1(q_i)) \to q_i}\,(\text{T-CONST}) \qquad \dfrac{}{\Gamma_2 \vdash x : q_1}\,(\text{T-VAR})}{\Gamma_2 \vdash bx : q_i}\,(\text{T-APP})$$

With this

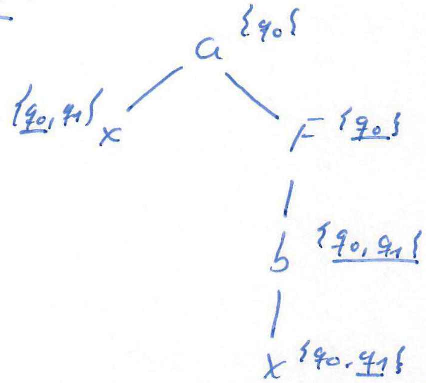
$$\dfrac{\dfrac{}{F : (\Theta, 2)^\varepsilon \vdash F : \Theta}\,(\text{T-VAR}) \qquad \dfrac{}{\Gamma_2 \vdash bx : q_0} \qquad \dfrac{}{\Gamma_2 \vdash bx : q_1}}{\Gamma_1 \vdash F(bx) : q_0}\,(\text{T-APP})$$

With this in mind we get

$$\dfrac{\dfrac{}{a : \theta_a}\,(\text{T-CONST}) \qquad \dfrac{}{x : (q_0, 2)^{\varepsilon} \vdash x : q_0}\,(\text{T-VAR}) \qquad \dfrac{}{\Gamma_1 \vdash F(bx) : q_0}}{F : (\theta, 2)^d,\ x : (q_0, 2)^{\varepsilon} \wedge (q_1, 2)^d \vdash a\, x\, (F(bx)) : q_0.}\,(\text{T-APP})$$

$$\dfrac{F : (\theta, 2)^d,\ x : (q_0, 2)^{\varepsilon} \wedge (q_1, 2)^d \vdash a\, x\, (F(bx)) : q_0}{F : (\theta, 2)^d \vdash \lambda x.\ a\, x\, (F(bx)) : \theta.}\,(\text{T-ABS})$$

Illustration:



Remark:

In (T-APP), $k = 0$ can be used,
meaning there are no requirements on the parameter.
For example, $x : (\top \to q,\ \Omega(q))^d \vdash x\, t : q$
is derivable for any t, even if t is ill-typed or contains variables other than x.

2.3 Typing Recursion Schemes

Goal: Define when a recursion scheme is well-typed, $\vdash_A G : q$

Problem: In programming languages, the rule for recursion $F = t$ is

$$\frac{\Gamma, F : \tau \vdash t : \tau}{\Gamma \vdash F : \tau.}$$

This only works for trivial acceptance: All / No infinite paths.

Solution: Define $\vdash_A G : q$ in terms of parity games.

Definition:

Consider APTA $A = (\Sigma, Q, \delta, q_I, \Omega)$

and scheme $G = (\Sigma, N, R, S)$.

We define the parity game $\mathcal{G}_{A,G} = (V_\forall, V_\exists, (S, q_I, \Omega(q_I)), E, \Omega')$

by

$$V_\exists = \{ (F, \Theta, m) \mid F \in dom(N), \ \Theta :: N(F) \}$$

$$V_\forall = \{ \Gamma \mid dom(\Gamma) \subseteq dom(N), \text{ all flags } f \}$$

$$E = \{ ((F, \Theta, m), \Gamma) \mid \Gamma \vdash R(F) : \Theta \}$$

$$\cup \{ (\Gamma, (F, \Theta, m)) \mid F : (\Theta, m)^f \in \Gamma \}.$$

The priority function Ω' maps
- (F, Θ, m) to m
- Γ to 0.

- G is well-typed, denoted by $\vdash_A G$,

if Player $\exists$ has a winning strategy in the parity game $\mathcal{G}_{A,G}$ (from the initial position).

Explanation:
- Player $\exists$ tries to prove that the recursion scheme is well-typed.

Player $\forall$ tries to disprove this.
- At node (F, Θ, m), Player $\exists$ has to pick a type environment Γ under which $R(F)$ has type Θ.

- Player $\forall$ then picks a binding $F' : (\Theta', m')^f$ from Γ and asks Player $\exists$ to show that F' has type Θ'.
- Then it is again Player $\exists$'s turn.
- The game continues indefinitely or ends if one of the players is unable to move.

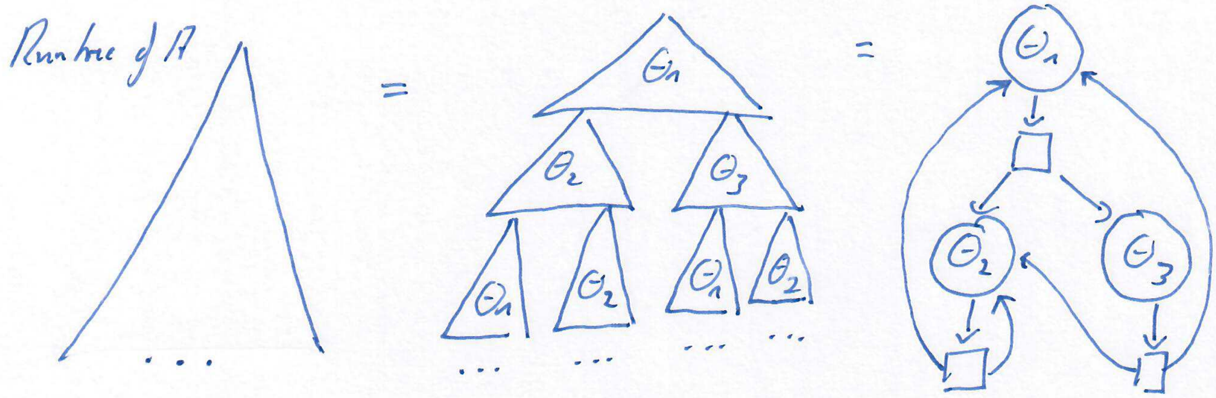
- Player ∃ wins if

  at some point the chosen type environment is empty (and thus Player ∀ cannot pick a binding)

  or the largest priority that occurs infinitely often is even.

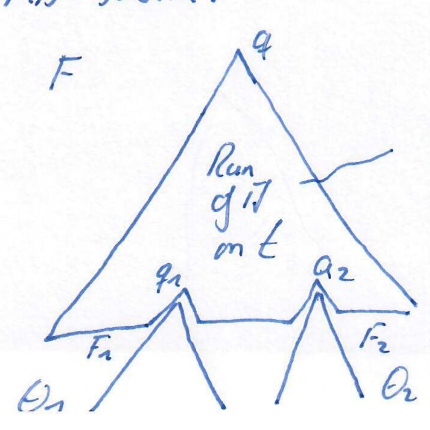
Intuition:

- Typing splits the runtree of the APTA into finitely many subtrees (after abstraction):



- More precisely, in (F, Θ, m),
  Θ serves as an abstraction for the subtree created by the right-hand side term $R(F)$.
  The subtree leads from leaves of $R(F)$ where non-terminals are called recursively to root F.
  The abstraction captures how the automaton changes its states on this subtree.

$F = t(F_1, F_2)$



Θ abstracts this part of the overall runtree.
What is the abstraction?
Behavior at the interface (no interference).

The behavior at the interface is Q,

say $(q_1, m_1) \to (q_2, m_2) \to q$.

- What is the role of m in (F, Q, m),

  or m_1 in (F_1, q_1, m_1)?

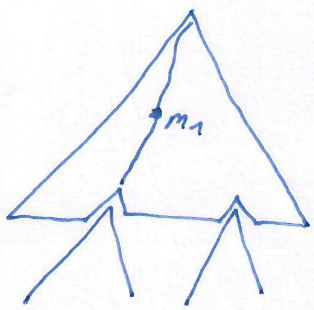
  This m_1 tracks the priority on the path through $R(F)$,

  in the above example from q to q_1.

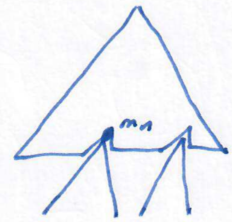
  This actually means we see the priority a bit later,

  namely when F_1 is called $\Rightarrow$ Does not matter in the infinite.

This is when m_1 occurs in the actual run tree



This is when we see m_1 in the abstraction



- One may be puzzled that Γ can change

  when having to type F several times.

  There is a unique best Γ to choose.

  It can be computed via a least fixed point.

  ↳ Larger Γ may make it impossible for Player $\exists$ to win.

  She could claim types that do not hold.

  ↳ Smaller Γ also may make it impossible for Player $\exists$ to win.

  Types could be missing that are needed in the computation.

Example:

Recall $G = (\Sigma, N, R, S)$ with $\Sigma = \{a : o \to o \to o,\ b : o \to o,\ c : o\}$

$N = \{S : o,\ F : o \to o\}$

$$R = \{ S \mapsto F\,c,\ F \mapsto \lambda x.\ a\ x\,(F(b\,x))\}.$$

and

$$\mathcal{M}_1 = (\Sigma,\ \{q_0, q_1\},\ \delta_1,\ q_0,\ \{q_0 \mapsto 2,\ q_1 \mapsto 1\})$$

where for all $q \in \{q_0, q_1\}$:

$$\delta_1(q, a) := (1, q) \wedge (2, q)$$

$$\delta_1(q, b) := (1, q_1)$$

$$\delta_1(q, c) := \mathrm{true}.$$

Let $\theta = (q_0, 2) \wedge (q_1, 2) \to q_0$.

Above, we showed the following type judgement:

$$F : (\theta, 2)^{\downarrow} \vdash \lambda x.\ a\ x\ (F(bx)) : \theta.$$

Moreover, one can check that

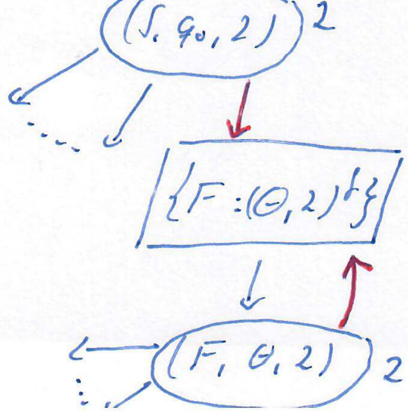
$$F : (\theta, 2)^{\downarrow} \vdash F\,c : q_0.$$

A memoryless winning strategy σ for the parity game $\mathcal{G}_{\mathcal{M}_1, G}$ is

$$\sigma(S, q_0, 2) = \{F : (\theta, 2)^{\downarrow}\}$$

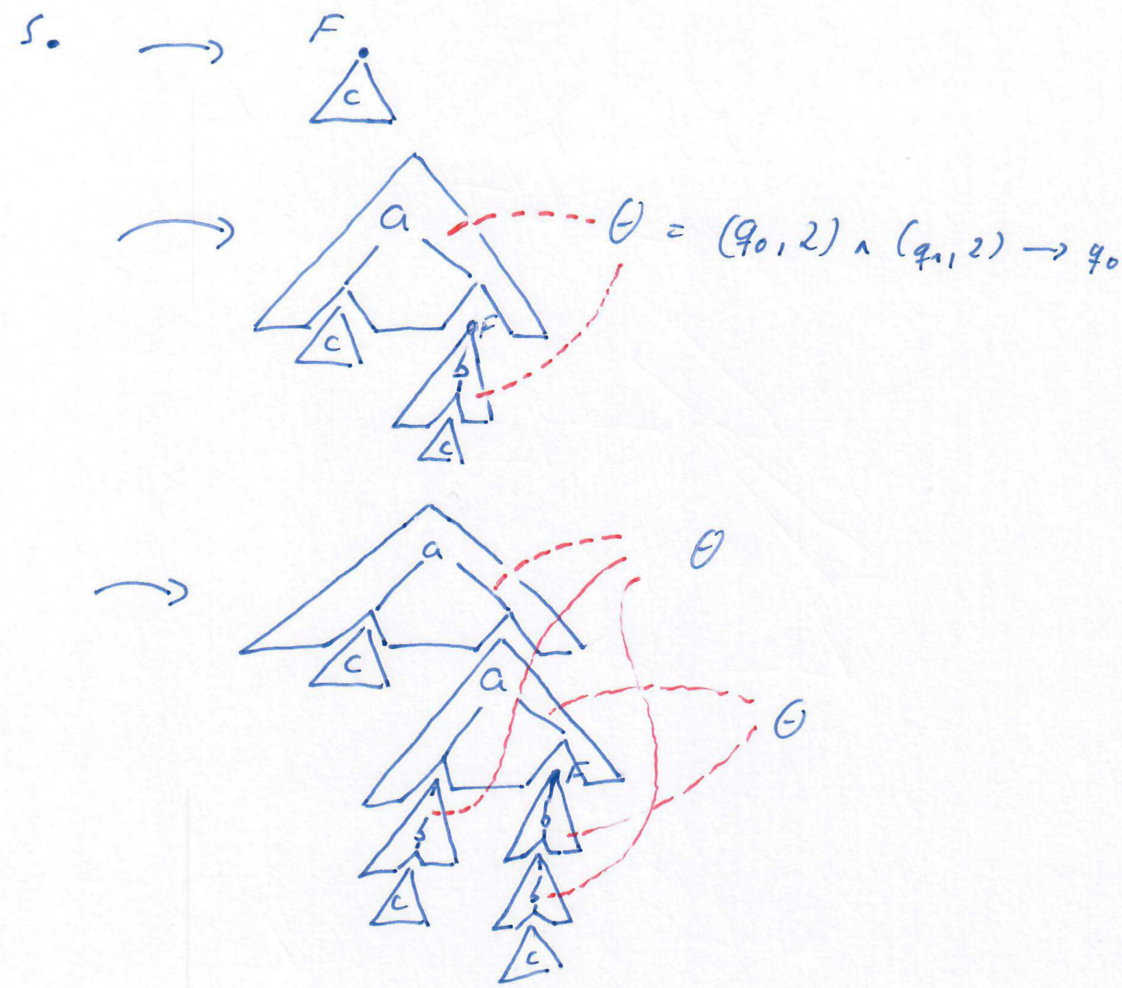
$$\sigma(F, \theta, 2) = \{F : (\theta, 2)^{\downarrow}\}.$$

Graphically:

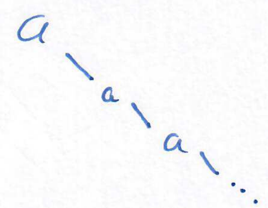


$\rightarrow\ := \sigma.$

Illustrated as finitely many subtrees:



Note: The path that is checked by the parity game is the one where F is rewritten infinitely often:



Still, we need the intersection type Θ to make sure finite parts of the computation also contribute to acceptance.
More precisely, the addition of b has to change the state in a way that is compatible with what a expects.
This information about finite computations, and thus Γ,

can be determined by a least fixed point.

Remark:

Note that the usual rule for recursion

$$\frac{\Gamma,\ F:(\theta,m)^f \vdash R(F):\theta}{\Gamma \vdash R(F):\theta}$$

and the definition

of $\vdash_A G$ by $\emptyset \vdash_A S:q_I$

is not sound.

Consider A_1' obtained from A_1

by replacing the initial state by q_1.

Let $G' = (\Sigma, \{S\}, \{S \mapsto b(S)\}, S)$.

Then $\emptyset \vdash S:q_1$

is derivable by

$$\dfrac{\dfrac{\emptyset \vdash b:(q_1,1)\to q_1 \qquad S:(q_1,1)^f \vdash S:q_1}{S:(q_1,1)^f \vdash b(S):q_1}}{\emptyset \vdash S:q_1}.$$

The tree $[\![G']\!]$ is, however, not accepted by A_1'.